DTx-EZ

Getting Started Manual

DTXEZ-903-01 Rev. A / January 2005

TEquipment.NET USA
An Interworld Highway, LLC Company

WARRANTY

Keithley Instruments, Inc. warrants this product to be free from defects in material and workmanship for a period of 3 years from date of shipment.

Keithley Instruments, Inc. warrants the following items for 90 days from the date of shipment: probes, cables, rechargeable batteries, diskettes, and documentation.

During the warranty period, we will, at our option, either repair or replace any product that proves to be defective.

To exercise this warranty, write or call your local Keithley representative, or contact Keithley headquarters in Cleveland, Ohio. You will be given prompt assistance and return instructions. Send the product, transportation prepaid, to the indicated service facility. Repairs will be made and the product returned, transportation prepaid. Repaired or replaced products are warranted for the balance of the original warranty period, or at least 90 days.

LIMITATION OF WARRANTY

This warranty does not apply to defects resulting from product modification without Keithley's express written consent, or misuse of any product or part. This warranty also does not apply to fuses, software, non-rechargeable batteries, damage from battery leakage, or problems arising from normal wear or failure to follow instructions.

THIS WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE. THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES.

NEITHER KEITHLEY INSTRUMENTS, INC. NOR ANY OF ITS EMPLOYEES SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF ITS INSTRUMENTS AND SOFTWARE EVEN IF KEITHLEY INSTRUMENTS, INC., HAS BEEN ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH DAMAGES. SUCH EXCLUDED DAMAGES SHALL INCLUDE, BUT ARE NOT LIMITED TO: COSTS OF REMOVAL AND INSTALLATION, LOSSES SUSTAINED AS THE RESULT OF INJURY TO ANY PERSON, OR DAMAGE TO PROPERTY.

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139
440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY (534-8453) • www.keithley.com

12/04

DTx-EZ
Getting Started Manual

©2005, Keithley Instruments, Inc.
All rights reserved.
First Printing, January 2005
Cleveland, Ohio, U.S.A.
Document Number: DTXEZ-903-01 Rev. A

Manual Print History

The print history shown below lists the printing dates of all Revisions and Addenda created for this manual. The Revision Level letter increases alphabetically as the manual undergoes subsequent updates. Addenda, which are released between Revisions, contain important change information that the user should incorporate immediately into the manual. Addenda are numbered sequentially. When a new Revision is created, all Addenda associated with the previous Revision of the manual are incorporated into the new Revision of the manual. Each new Revision includes a revised copy of this print history page.

Revision A (Document Number DTXEZ-903-01A) ..... January 2005

All Keithley product names are trademarks or registered trademarks of Keithley Instruments, Inc.
Other brand and product names are trademarks or registered trademarks of their respective holders.

KEITHLEY

Safety Precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with non-hazardous voltages, there are situations where hazardous conditions may be present.

This product is intended for use by qualified personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product. Refer to the manual for complete product specifications.

If the product is used in a manner not specified, the protection provided by the product may be impaired.

The types of product users are:

Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring that operators are adequately trained.

Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.

Maintenance personnel perform routine procedures on the product to keep it operating properly, for example, setting the line voltage or replacing consumable materials. Maintenance procedures are described in the manual. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.

Service personnel are trained to work on live circuits, and perform safe installations and repairs of products. Only properly trained service personnel may perform installation and service procedures.

Keithley products are designed for use with electrical signals that are rated Measurement Category I and Measurement Category II, as described in the International Electrotechnical Commission (IEC) Standard IEC 60664. Most measurement, control, and data I/O signals are Measurement Category I and must not be directly connected to mains voltage or to voltage sources with high transient over-voltages. Measurement Category II connections require protection for high transient over-voltages often associated with local AC mains connections. Assume all measurement, control, and data I/O connections are for connection to Category I sources unless otherwise marked or described in the Manual.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30V RMS, 42.4V peak, or 60VDC are present. **A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.**

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000 volts, **no conductive part of the circuit may be exposed.**

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

Before operating an instrument, make sure the line cord is connected to a properly grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

When installing equipment where access to the main power cord is restricted, such as rack mounting, a separate main input power disconnect device must be provided, in close proximity to the equipment and within easy reach of the operator.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

The instrument and accessories must be used in accordance with its specifications and operating instructions or the safety of the equipment may be impaired.

Do not exceed the maximum signal levels of the instruments and accessories, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.

When fuses are used in a product, replace with same type and rating for continued protection against fire hazard.

Chassis connections must only be used as shield connections for measuring circuits, NOT as safety earth ground connections.

If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

If a screw is present, connect it to safety earth ground using the wire recommended in the user documentation.

The symbol on an instrument indicates that the user should refer to the operating instructions located in the manual.

The symbol on an instrument shows that it can source or measure 1000 volts or more, including the combined effect of normal and common mode voltages. Use standard safety precautions to avoid personal contact with these voltages.

The symbol indicates a connection terminal to the equipment frame.

The **WARNING** heading in a manual explains dangers that might result in personal injury or death. Always read the associated information very carefully before performing the indicated procedure.

The **CAUTION** heading in a manual explains hazards that could damage the instrument. Such damage may invalidate the warranty.

Instrumentation and accessories shall not be connected to humans.

Before performing any maintenance, disconnect the line cord and all test cables.

To maintain protection from electric shock and fire, replacement components in mains circuits, including the power transformer, test leads, and input jacks, must be purchased from Keithley Instruments. Standard fuses, with applicable national safety approvals, may be used if the rating and type are the same. Other components that are not safety related may be purchased from other suppliers as long as they are equivalent to the original component. (Note that selected parts should be purchased only through Keithley Instruments to maintain accuracy and functionality of the product.) If you are unsure about the applicability of a replacement component, call a Keithley Instruments office for information.

To clean an instrument, use a damp cloth or mild, water based cleaner. Clean the exterior of the instrument only. Do not apply cleaner directly to the instrument or allow liquids to enter or spill on the instrument. Products that consist of a circuit board with no case or chassis (e.g., data acquisition board for installation into a computer) should never require cleaning if handled according to instructions. If the board becomes contaminated and operation is affected, the board should be returned to the factory for proper cleaning/servicing.

Table of Contents

About this Manual xiii
Intended Audience xiii
What You Should Learn from this Manual xiii
Organization of this Manual xiv
Conventions Used in this Manual xiv
Related Information xv
Where to Get Help xvi

Chapter 1: Overview 1
What is DTx-EZ? 2
Conforms to the DT-Open Layers Standard 2
Provides Custom Controls 2
The Data Acquisition Custom Control 3
The DT Plot Custom Control 3
Provides Properties, Methods, and Events 4
Provides Function and Subroutine Libraries 5
Follows Object-Oriented Design 6
Provides Extensive Example Programs 6
Provides Multiple Board Support 6
Provides High Performance 7
What You Need 8
Installation 9
Adding DTx-EZ Custom Controls to Your Project 10
Adding to a Visual Basic 6.0 Project 10
Adding to a Visual C++ 6.0 Project 11
Creating an Application 12
Using the DTx-EZ Online Help 13

Chapter 2: Using the DTx-EZ Examples 15
About the Examples 16
Running the Examples as Applications 17
Opening the Examples from within Visual Basic 18
Opening the Examples from within Visual C++ 19
A/D Burst Example 20
DAC Waveform Generator Example 25
Continuous A/D Example 29
Single-Value Example 31
About-Trigger Example 33
Digital I/O Example 35
DDE Server and Client Examples 37
Waveform Generator Example 39
Continuous FFT Example 40
ChartIt Example 41
Scope Example 42

Chapter 3: Property, Method, Function, and Subroutine Summary 45
Introduction 46
Data Acquisition Custom Control 47
Information Properties and Methods 47
Initialization Properties 57
Configuration Properties and Functions 58
Operation Properties, Methods, Functions, and Subroutines 61
Data Management DLLs 64
Buffer Management Functions and Subroutines 64
Conversion Functions and Subroutines 67
DT Plot Custom Control 69
Plot Appearance 69

Plot Pre-Display Operational Parameters 70
Grids 71
Markers 71
x-Axis Parameters 72
y-Axis Parameters 73
Plotting Operation Control Parameters 73

Chapter 4: Programming Flowcharts 75
Introduction 76
Single-Value Operations 77
Continuous Buffered Input Operations 79
Continuous Buffered Output Operations 81
Event Counting Operations 83
Frequency Measurement Operations 85
Pulse Output Operations 87
Plotting Control Operations 89

Chapter 5: Software Architecture 91
Introduction 92
System Operations 93
Initializing and Specifying a Board 93
Specifying a Subsystem 94
Configuring a Subsystem 95
Handling Events 96
Handling Errors 96
Halting the Operation 98
Analog and Digital I/O Operations 99
Data Encoding 99
Resolution 100
Channels 100
Specifying the Channel Type 101

Specifying a Single Channel 102
Specifying One or More Channels 102
Specifying the Channel List Size 103
Specifying the Channels in the Channel List 104
Inhibiting Channels in the Channel List 105
Specifying Synchronous Digital I/O Values in the Channel List 106
Ranges 108
Gains 109
Specifying the Gain for a Single Channel 109
Specifying the Gain for One or More Channels 109
Filters 111
Data Flow Modes 112
Single-Value Operations 112
Continuous Operations 113
Continuous (Post-Trigger) Mode 114
Continuous Pre-Trigger Mode 115
Continuous About-Trigger Mode 116
Triggered Scan Mode 118
Scan-Per-Trigger Mode 119
Internal Retrigger Mode 120
Retrigger Extra Mode 121
Clock Sources 122
Internal Clock Source 122
External Clock Source 123
Extra Clock Source 124
Trigger Sources 124
Software (Internal) Trigger Source 125
External Digital (TTL) Trigger Source 125
External Analog Threshold (Positive) Trigger Source 126

External Analog Threshold (Negative) Trigger Source 126
Analog Event Trigger Source 127
Digital Event Trigger Source 127
Timer Event Trigger Source 127
Extra Trigger Source 127
Buffers 128
Ready Queue 128
Done Queue 131
Buffer and Queue Management 133
Buffer Wrap Modes 134
DMA Resources 135
Counter/Timer Operations 137
Counter/Timer Operation Mode 138
Event Counting 138
Frequency Measurement 140
Using the Windows Timer 140
Using a Pulse of a Known Duration 142
Rate Generation 144
One-Shot 148
Repetitive One-Shot 151
C/T Clock Sources 154
Internal C/T Clock 155
External C/T Clock 155
Internally Cascaded Clock 156
Extra C/T Clock Source 157
Gate Types 157
Software Gate Type 158
High-Level Gate Type 158
Low-Level Gate Type 158

Low-Edge Gate Type 159
High-Edge Gate Type 159
Any Level Gate Type 159
High-Level, Debounced Gate Type 160
Low-Level, Debounced Gate Type 160
High-Edge, Debounced Gate Type 160
Low-Edge, Debounced Gate Type 161
Level, Debounced Gate Type 161
Pulse Output Types and Duty Cycles 162
Simultaneous Operations 164
Plot Control Operations. 166
Plotting Data 166
Data Identification Properties 166
Plotting Mechanics Properties 167
Appearance 167
Stripchart Mode 168
Specifying a Grid. 168
Specifying Markers 169

Appendix A: Flowcharts for Substeps 171

Chapter 6: Product Support 191
General Checklist 192
Service and Support. 193

Index 195

About this Manual

This manual describes how to get started using DTx-EZ™ to develop application programs for data acquisition boards that conform to the DT-Open Layers™ standard.

Intended Audience

This document is intended for engineers, scientists, technicians, or others responsible for developing application programs using the Microsoft Visual Basic™ or Visual C++™ compiler to perform data acquisition operations.

It is assumed that you are a proficient programmer, that you are experienced programming in the Windows® 98, Windows NT, Windows Me (Millennium Edition), or Windows 2000 operating environment on the IBM PC or compatible computer platform, that you have familiarity with data acquisition principles, and that you have clearly defined your requirements.

What You Should Learn from this Manual

This manual provides installation instructions, summarizes the functions provided by the DTx-EZ, and describes how to use the properties, methods, functions, and subroutines to develop a data acquisition program. Using this manual, you should be able to successfully install the DTx-EZ software and get started writing an application program for data acquisition.

This manual is intended to be used with the online help for the DTx-EZ, which you can find in the same program group as the DTx-EZ software. The online help for the DTx-EZ contains all of the specific reference information for each of the properties, methods, functions, subroutines, error codes, and Windows messages (events).

Organization of this Manual

This manual is organized as follows:

- Chapter 1, "Overview," provides an overview of DTx-EZ.
- Chapter 2, "Using the DTx-EZ Examples," describes how to use the example programs provided with DTx-EZ.
- Chapter 3, "Property, Method, Function, and Subroutine Summary," summarizes the properties, methods, functions, and subroutines provided with DTx-EZ.
- Chapter 4, "Programming Flowcharts," provides programming flowcharts for using the properties, methods, functions, and subroutines provided with DTx-EZ.
- Chapter 5, "Software Architecture," describes the architecture and concepts of DTx-EZ software.
- Chapter 6, "Product Support," describes how to get help if you have trouble using DTx-EZ.
- Appendix A, "Flowcharts for Substeps," provides additional flowcharts for performing substeps required for an operation.
- An index completes this document.

Conventions Used in this Manual

The following conventions are used in this manual:

- Notes provide useful information that requires special emphasis, cautions provide information to help you avoid losing data or damaging your equipment, and warnings provide information to help you avoid catastrophic damage to yourself or your equipment.
- Items that you select or type are shown in **bold**. Property, method, function, and subroutine names also appear in bold.
- Code fragments appear in `courier font`.

- In syntax, items inside square brackets are optional.
- In syntax, a vertical bar between braces indicates that you must choose between two or more items. You must choose an item unless all of the items are also enclosed in curly brackets. For example, { True | False } indicates that you must select one of these choices.
- When navigating the screens, an instruction such as **Configure > Board** means to select "Board" from the drop-down menu under "Configure."

Related Information

Refer to the following documentation for more information on using DTx-EZ:

- DTx-EZ Online Help. This Windows help file is located in the same program group as the DTx-EZ software and contains all of the specific reference information for each of the properties, methods, subroutines, functions, error codes, and Windows messages (events) provided by DTx-EZ. Refer to page 13 for information on how to launch this help file.
- Device Driver documentation for your board. This documentation describes the capabilities supported by the device driver for your board. Refer to this documentation as you develop application programs using DTx-EZ.
- User manual for your data acquisition board. This manual describes the capabilities of the hardware as well as how to set up and install your board.
- For Visual Basic and Visual C++ programmers, see the online help in your programming environment.
- Windows programming documentation.

Where to Get Help

Should you run into problems installing or using DTx-EZ, the Keithley Technical Support Department is available to provide technical assistance.

Overview

What is DTx-EZ? 2
What You Need 8
Installation 9
Adding DTx-EZ Custom Controls to Your Project 10
Creating an Application 12
Using the DTx-EZ Online Help 13

What is DTx-EZ?

DTx-EZ is a set of Object-Linking and Embedding (OLE) controls that facilitate rapid data acquisition application development in Visual Basic and Visual C++ environments.

DTx-EZ supplies the programming tools to transform Visual Basic and Visual C++ into a powerful data acquisition application development environment. With DTx-EZ, you can quickly develop applications with analog and digital I/O, real-time data display, and a graphical user interface (GUI), while operating in a true Windows environment.

Conforms to the DT-Open Layers Standard

DTx-EZ is fully compatible with DT-Open Layers.

DT-Open Layers is a set of open standards for developing integrated, modular software under Microsoft Windows. Because it is modular and uses Windows DLLs, DT-Open Layers is easily expanded to support new, more powerful hardware devices without re-linking or rebuilding applications. Therefore, you do not need to rewrite your code when adding new data acquisition boards that have DT-Open Layers-compliant device drivers. DT-Open Layers protects your software investment now and in the future.

Provides Custom Controls

The DTx-EZ provides two custom controls:

- Data Acquisition Custom Control, and
- Plotting Custom Control.

The following subsections describe these controls.

The Data Acquisition Custom Control

The Data Acquisition Custom Control facilitates performing data acquisition functions. Each copy of the Data Acquisition Custom Control operates on a single subsystem of the supported board at a time. However, Visual Basic and Visual C++ lets you use multiple copies of this control operating simultaneously, controlling different subsystems on the same board or even on different boards.

DTx-EZ determines the capabilities of each subsystem for your data acquisition board. The board's supported capabilities are listed in the custom control's Properties window. You can control the subsystem's operation by manipulating the subsystem's properties. You can change the properties at design time in the Properties window or at run time using simple Visual Basic or Visual C++ code.

Each subsystem may have multiple channels. For example, the A/D subsystem on the DataAcq-EZ boards has 16 available channels. To access multiple channels, you must set up a list of channels you want to sample. The DataAcq-EZ boards also provide programmable gains. To program the gains on the channels, you need to set up a gain list to accompany the channel list.

The DT Plot Custom Control

The DT Plot Custom Control is a high-speed plotting control, useful for plotting fixed- or floating-point data in your Visual Basic or Visual C++ application. Since the DT Plot Custom Control works directly with DT-Open Layers *hBuf* data, the need to copy the buffer to an array is eliminated.

You fill a buffer with data you want to plot and assign it to the plot's **Buffer** property. You can plot from 1 to 16 channels of data at a time, all with different colors. You may also choose from a variety of line styles and the following features:

- x and y grid lines that you can set to automatically scale to your data or to remain at fixed spacing intervals.
- x and y markers that can be used to indicate a zoom-in section of the data or to determine the data value at that position.
- A stripchart mode so you can plot many buffers of rapidly-changing, continuous data and see all of them on the plot's display.
- A single-point feature when in strip charting mode that allows you to add one point of data at a time to the display.

At design time, the DT Plot Custom Control displays a randomly-generated data plot that shows you what your plots will look like. Whenever you change a property, the plot immediately displays the effects of your changes. For some properties, you can enter new data for the plot at either design time or run time.

Provides Properties, Methods, and Events

The Data Acquisition and DT Plot Custom Controls have unique *properties, methods,* and *events*, described as follows:

- ***Properties*** represent the variables that allow you to configure the data acquisition or plotting operation. Many read/write properties are accessible in the Properties window at design time or at runtime. Some read/write properties are lists that may be accessed through the control's custom property pages. Other properties are read-only; you can access them only at runtime. These read-only properties represent variables that you can use to determine data acquisition or plotting capabilities.

Note: Each Data Acquisition Custom Control can be associated with a single subsystem at a time. *This means that you must first select the board and subsystem you want to use before you can configure the subsystem's properties.* When a custom control is created, its properties are set to the default settings. You must modify the properties if you want to change their values from the default settings.

- ***Methods*** are tools that are used in the Visual Basic or Visual C++ code to provide runtime control of data acquisition operations; no methods are associated with the DT Plot Custom Control.
- ***Events*** have procedures that execute code when the specified data acquisition event occurs during runtime in the case of the Data Acquisition Custom Control or when the specified mouse or keyboard event occurs during runtime in the case of the DT Plot Custom Control. Most OLE custom control events are based on user interactions.

  Note, however, that the Data Acquisition Custom Control events are based on specific data acquisition events that could occur within your application. Most of the Data Acquisition Custom Control events are used for either continuous acquisition operation or for error tracking.

  All of the DT Plot Custom Control events are standard Microsoft events. Refer to your Visual Basic or Visual C++ online help for more information about them.

Provides Function and Subroutine Libraries

In addition to custom controls, DTx-EZ provides a library of DT-Open Layers functions and subroutines for Visual Basic and for Visual C++. These libraries add facilities for managing buffers, simultaneously starting multiple subsystems, and performing FFT analysis.

Follows Object-Oriented Design

For easy programming, DTx-EZ's Application Programming Interface (API) emphasizes polymorphism — it uses nearly identical interface functions to communicate with each type of data acquisition subsystem: analog input (A/D), analog output (D/A), digital input (DIN), digital output (DOUT), and counter/timer (C/T).

The features provided by each supported data acquisition board vary; for a complete list of capabilities supported by your board, refer to board's driver documentation.

The API provides a full set of functions to query and set all possible device capabilities. The library hides device details and presents a consistent interface to each subsystem.

Provides Extensive Example Programs

To get your application up and running quickly, a comprehensive set of Visual Basic and Visual C++ example programs is provided. You can use these examples as tutorials to learn how DTx-EZ operates, or you can modify one or more examples to form the basis of your own custom data acquisition application. Source code is included, so you can customize the examples to complete your Visual Basic or Visual C++ application.

Provides Multiple Board Support

The DTx-EZ is hardware-independent. You can add support for new boards without altering or recompiling code at the application level simply by adding a new DT-Open Layers device driver. You install the device driver separately (in the Windows environment); refer to your board and/or device driver documentation for more information on installing device drivers.

The library functions are designed to fully support all board features.

Provides High Performance

DTx-EZ was designed with an intimate knowledge of the Windows operating environment and the IBM PC computer system. As a result, it takes unique advantage of the Windows architecture to achieve maximum performance. By using sophisticated software buffering (part of the DT-Open Layers standard), and the PCI bus, USB bus, or DMA capabilities of the hardware, the software can achieve continuous throughput to or from memory at greater than 1 MHz.

What You Need

To use DTx-EZ, you need the following:

- Pentium or higher-based PC with a CD-ROM drive and a minimum of 32 Mbytes of RAM;
- One or more supported data acquisition boards;
- Microsoft Windows 2000 or XP.
- Microsoft Visual C++ 6.0 or Microsoft Visual Basic 6.0.

Installation

DTx-EZ is installed automatically when you install the device driver for the module. Refer to the getting started manual for your module for more information.

Adding DTx-EZ Custom Controls to Your Project

The following subsections describe how to add DTx-EZ Custom Controls to:

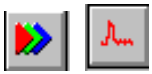
- A Visual Basic 6.0 project (this page),
- A Visual C++ 6.0 project (page 11).

Adding to a Visual Basic 6.0 Project

Before you begin using the DT-EZ, add the DTx-EZ Data Acquisition and DT Plotting Custom Controls and definition files. Add the files DTACQ32.OCX, DTPLOT32.OCX, OLMEMDEFS.BAS, OLDADEFS.BAS and OLDSPDEFS.BAS to your project file as follows from Visual Basic 6.0:

1. Select **Project** > **Components**.
   The Components dialog box appears.
2. Click the **Controls** tab.
3. Click **DTAcq32 OLE Custom Control module** and/or **DTPlot32 OLE Custom Control module**, then click **OK**.
4. Choose **Project > Add Module**.
   The Add Module dialog box appears.
5. Select the **Existing** tab.
6. Select **OLMEMDEFS.BAS**, **OLDEFS.BAS**, and/or **OLDSPDEFS.BAS** from the \DTx-EZ\INCLUDE directory, and click **Open**.

When the DTx-EZ custom controls are loaded, the DTACQ32 and DTPLOT32 icons appear in the Toolbox as follows:



Adding to a Visual C++ 6.0 Project

Before you begin using the DT-EZ, add the DTx-EZ Data Acquisition and DT Plotting Custom Controls.

1. Start Windows 95 and launch Microsoft Visual C++ 6.0.
2. Create a new MFC project using the application wizard.
   Ensure you add support for ActiveX controls in Step 2.
3. Select **Project > Add to Project > Components and Controls**.
   The "Component and Controls Gallery" appears.
4. Double-click the **Registered ActiveX Controls** folder.
5. Select the **DTPlot32 Control** and click **Insert**.
   The program prompts you to insert the component and to confirm the creation of a wrapper class for accessing the control through Visual C++.
6. Repeat step 5 for the **DTAcq32 Control**.

Creating an Application

You can use the Data Acquisition Custom Control and DT Plot Custom Control just like other ActiveX or OLE custom controls to integrate data acquisition into your Windows application. Create your application as follows:

1. Add the Data Acquisition or DT Plot Custom Control to your form by selecting it from the toolbox and placing it on your form. *For information on adding a DTx-EZ Custom Control to a project, see page 10. Once added to your project, you can select the DTx-EZ Custom Control from the toolbox.*

2. Configure your control object for the desired function by setting values in the Properties window.
   Refer to Chapter 5, "Software Architecture," for more information. Context-sensitive online help is available for the DTx-EZ Custom Controls as well. To access online help, simply press F1, and information related to the current operation appears on the screen.

3. Add code, as needed, using the Code window to

   – Respond to user actions,

   – Change the properties at run time, or

   – Control data acquisition operations.

   Refer to Chapter 2, "Using the DTx-EZ Examples," for more information.

4. When you are ready to run your application outside your development environment, create an executable file (.EXE) by choosing **Make EXE** file from the File menu.

Note: When you create and distribute applications that use the Data Acquisition Custom Control and DT Plot Custom Control, review the licensing material included in the DTx-EZ online help.

Using the DTx-EZ Online Help

This manual is intended to be used with the online help for DTx-EZ. The online help contains all of the specific reference information for each of the functions, error codes, and Windows messages not included in this manual.

To launch this online help, double-click the DTx-EZ help icon in the KUSB Series program group or folder.

2

Using the DTx-EZ Examples

About the Examples 16
A/D Burst Example 20
DAC Waveform Generator Example 25
Continuous A/D Example 29
Single-Value Example 31
About-Trigger Example 33
Digital I/O Example 35
DDE Server and Client Examples 37
Waveform Generator Example 39
Continuous FFT Example 40
ChartIt Example 41
Scope Example 42

About the Examples

DTx-EZ provides the software tools to create Visual Basic and Visual C++ data acquisition applications quickly and easily. A comprehensive set of examples shows you how to use DTx-EZ's Data Acquisition and DT Plotting Custom Controls in the Visual Basic or Visual C++ environment. If your needs are simple, choose one of the example programs; DTx-EZ will perform data acquisition right out of the box. Since all example source code is included, you can easily modify the examples to suit your needs, combine two or more examples, or extend the examples with your own code.

The following examples are provided:

- **A/D Burst Example** –Acquires data to disk; demonstrates channel-gain list setup (page 20).
- **DAC Waveform Generator Example** –Generates sine, square, or triangle output waveforms. It also demonstrates “zooming in” using the plot control (page 25).
- **Continuous A/D Example** –Continuously samples and displays multiple data points (page 29).
- **Single Value Example** –Acquires a single value from an A/D subsystem; outputs a single value to a D/A subsystem (page 31).
- **About-Trigger Example** –Acquires data to disk from initialization until the trigger event occurs and then for one second after (page 33).
- **Digital I/O Example** –Controls digital I/O lines (page 35).
- **DDE Server and Client Example** –Moves data to or from other applications using Window's Dynamic Data Exchange (page 37).
- **Waveform Generator Example** –Demonstrates the use of counter/timers to generate square waves (page 39).
- **Continuous FFT Example** –Computes and displays FFTs of the input data (page 40).

- **ChartIt Example** –Demonstrates how to use the stripchart mode to display single points of data (page 41).
- **Scope Example** –Generates and displays analog input channels including typical oscilloscope functions (page 42).

This chapter explains how to use each example. To see actual signals being acquired, connect a signal source to the analog inputs of your data acquisition board. (You can choose to run the examples without connecting a signal source to the board.)

You can run the examples as applications. However, if you wish to view or modify the source code for the examples, open the associated .VBP files from within Visual Basic (see page 18), or open the associated .MDP files from within Visual C++ (see page 19).

Note: Before running the examples, make sure the device driver for your data acquisition board has been installed.

Running the Examples as Applications

If you wish to use them as tutorials to learn how DTx-EZ operates, you can run the examples as applications as follows:

1. Start your operating system.
2. In the start menu, select the icon for the desired example from the DTx-EZ program group.

Opening the Examples from within Visual Basic

To customize the DTx-EZ example code for your own application, you must open the example's build file, .VBP, from within Visual Basic as follows:

1. Start your operating system, and start Visual Basic.
2. Choose **File > Open Project**.
3. Select the examples from the directory DTx-EZ\examples\vb. (The examples have a .VBP extension.)

Table 1: Visual Basic Example Programs and Their .VBP Files

Example	Directory
A/D Burst	\adburst\adburst.vbp
DAC Waveform	\dacwave\dacwave.vbp
Continuous A/D	\contdisp\contdisp.vbp
Single value	\sv\sv.vbp
Digital I/O	\dio\dio.vbp
DDE Server	\dde\server.vbp
DDE Client	\dde\client.vbp
Wave Generator	\wavegen\wavegen.vbp
About-Trigger	\abouttrigger\abouttrigger.vbp
Continuous FFT	\contfft\contfft.vbp
ChartIt	\chartit\chartit.vbp
Scope	\scope\scope.vbp

4. Select **Run > Start** to run each example.

Opening the Examples from within Visual C++

To customize the DTx-EZ example code for your own application, you must open the example's build file, .MDP, from within Visual C++ as follows:

1. Start your operating system, and start Visual C++.
2. Choose **File > Open Workspace**.
3. Select the examples from the directory DTx-EZ\examples\cpp. (The examples have a .MDP extension.)

Table 2: Visual C++ Example Programs and Their .MDP Files

Example	Directory
A/D Burst	\adburst\adburst.mdp
DAC Waveform	\dacwave\dacwave.mdp
Continuous A/D	\contdisp\contdisp.mdp
Single value	\sv\sv.mdp
Digital I/O	\dio\dio.mdp
DDE Server	\dde\server\server.mdp
DDE Client	\dde\client\client.mdp
Wave Generator	\wavegen\wavegen.mdp
About-Trigger	\abouttrigger\abouttrigger.mdp
Continuous FFT	\contfft\contfft.mdp
ChartIt	\chartit\chartit.mdp
Scope	\scope\scope.mdp

4. Select **Run > Start** to run each example.

A/D Burst Example

This example (adburst.vbp or adburst.mdp) samples multiple analog input channels and places the resulting data in a disk file for archiving and post-acquisition analysis. This example also demonstrates how to configure a channel-gain list and the A/D subsystem for a data acquisition board. You could use this example once your application is debugged to store actual data values.

When you run the example, the "Select Board" dialog box appears. The A/D Buffer To File screen is shown in Figure 1.

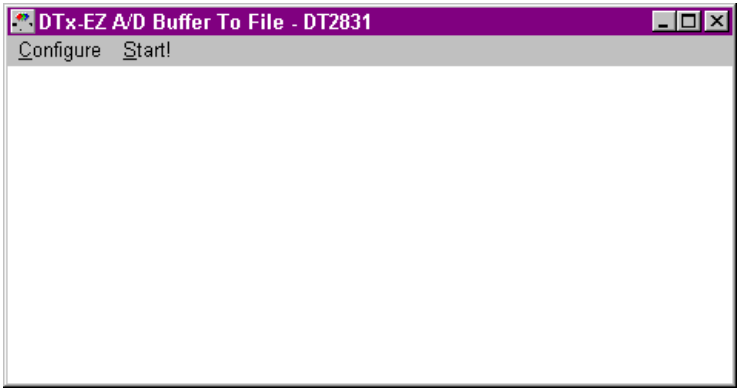


Figure 1: A/D Buffer To File Screen

Follow these steps:

1. Select **Configure > Board** to select a board from the list.
2. Select **Configure > CGL** to set up your board's channel-gain list. *The Channel/Gain List Setup dialog box, shown in Figure 2, appears.*

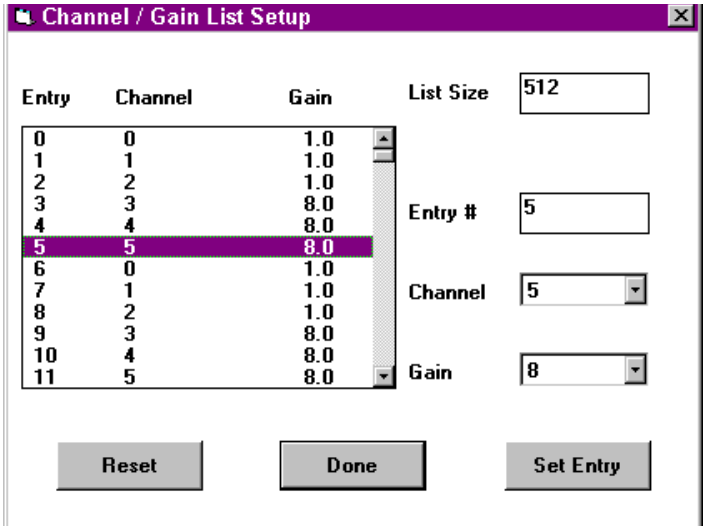


Figure 2: Setting Up a Channel-Gain List

During data acquisition, the channel-gain list automatically selects channel and gain values without compromising throughput. You can configure the list with channel numbers and associated gains.

To set up the channel-gain list, perform the following steps:

a. Select the list size, and then choose channel and gain values for each entry in the list.

b. After changing each item, click **Set Entry** to confirm the setting.

c. When you have completed the setup, click **Done** to return to the main menu.

The screen shown in Figure 2 illustrates a 512-entry channel-gain list that repeatedly scans channels 0 through 5 using a gain of 1 for channels 0-2 and a gain of 8 for channels 3-5.

3. Select **Configure > Input** to configure the analog input settings for your data acquisition board.
The Input Options dialog box, shown in Figure 3, appears.

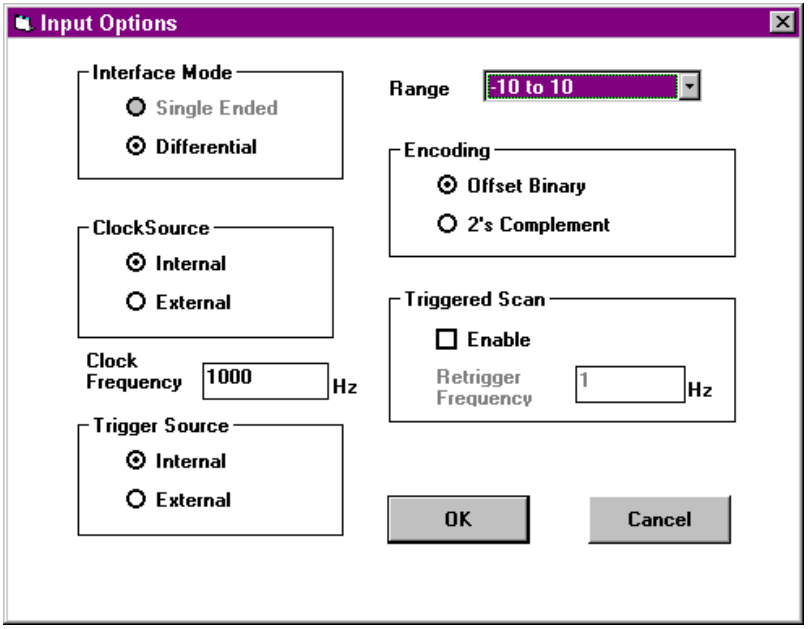


Figure 3: Configure the Analog Input Settings

Any settings that are not software-configurable for your board are inactive. The available settings are as follows:

– **Interface Mode —** allows you to select either single-ended or differential inputs. Most data acquisition boards can be configured for either 16 single-ended or 8 differential input channels. *Single-ended* inputs share a common ground. *Differential* inputs use a separate ground for each channel (which halves the channel capacity). Differential inputs can improve accuracy where long cables, low-level input ranges (< 1 V full-scale), or high resolution converters (> 12 bits) are used.

Note: If your board uses an onboard jumper to set the input mode, do not change this setting until you remove the board and change the jumper configuration to correspond with the new selection. If your board provides software-configurable input mode selection, you do not need to change any jumper settings.

– **Clock Source** and **Trigger Source** — You can select either an internal or an external **clock source** and **trigger source**. Clock frequency sources and triggers help you synchronize data conversions with off-board events. External frequency sources can also be used to produce clock frequencies that cannot be achieved with the board's onboard oscillator.

– **Range** — allows you to select the input voltage range. Ranges can be unipolar or bipolar.

– **Encoding** — lets you choose the input data encoding format.

Note: Because older boards use an onboard jumper to set the input voltage range and data encoding format, these settings cannot be changed until you remove the board and change the jumper configuration to correspond with the new selections. (If your board provides software-configurable settings, you do not need to change any jumper settings.)

– **Triggered Scan —** You can enable this mode on boards that support this feature. (Refer to your board's driver documentation to determine if it is supported on your board.) Triggered scan mode performs scans through the channel-gain list, where each scan is initiated by the onboard trigger. On some boards, the interval between the scans is programmable.

4. Select the desired options and click **OK**.
5. When you have completed the configuration, select **Start!** to begin acquiring data.
   *Note that the menu name changes to **Stop!** until the operation is done. You can choose **Stop!** to halt the operation at any time.*

   When the acquisition is complete, a message indicates that a buffer of data was collected and where the file was created.

DAC Waveform Generator Example

The DAC (digital-to-analog converter) waveform generator example (dacwave.vbp or dacwave.mdp) uses the D/A subsystem to continuously output a sine wave, square wave, or triangle wave. You can specify waveform frequency and the board on which to output the signal. You could use this example to supply a test signal for circuit evaluation or a stimulus to your experiment.

When you run the example, the **Select Board** dialog box appears. Choose the desired board from the list and click **OK**. The DAC Waveform Generator screen, shown in Figure 4, appears.

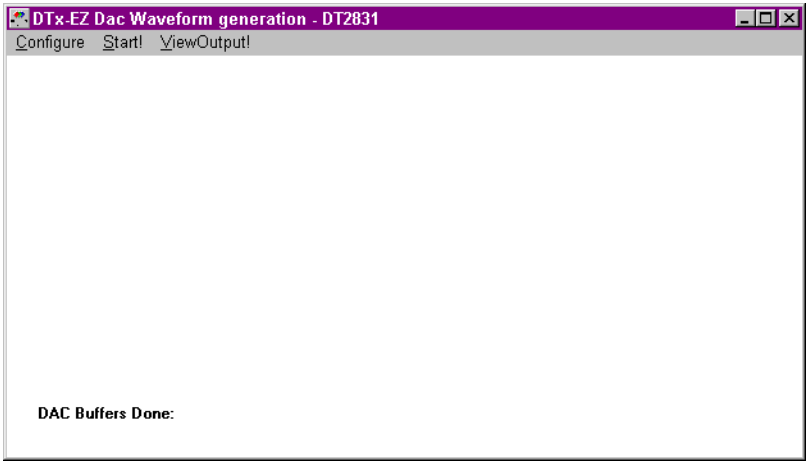


Figure 4: DAC Waveform Generator Screen

Follow these steps:

1. Select **Configure > Board** to choose a data acquisition board from the list.
2. Select **Configure > Output** to configure the analog output settings for your board.
   The Output Options dialog box, shown in Figure 5, appears. Refer to the A/D Burst example in the previous section for information on these settings.

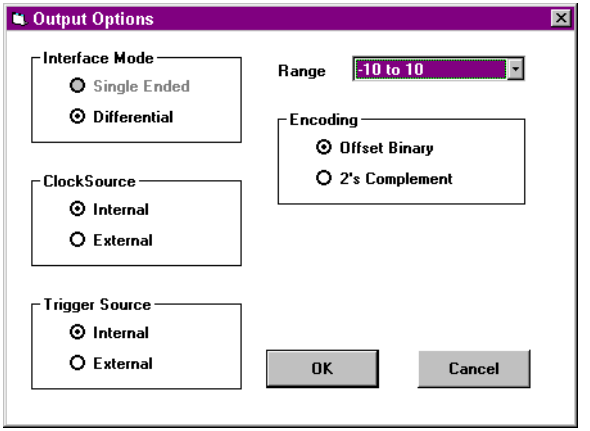


Figure 5: Configure the Analog Output Settings

3. Select **Configure > Acquisition** to configure your output waveform.
   The Acquisition Options dialog box, shown in Figure 6, appears.

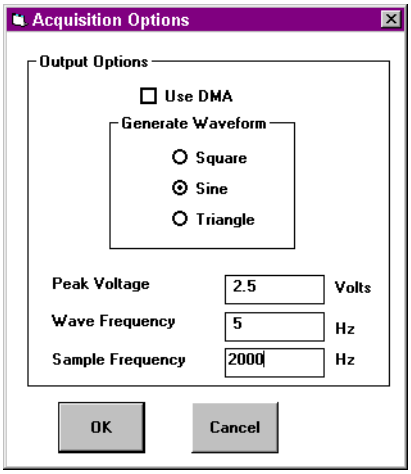


Figure 6: Configure Output Waveform

- Click **Use DMA** to enable direct memory access (DMA) for data transfer operations. (For optimum speed, DMA should be enabled on boards that support it.)
- Select the **type of waveform** you wish to produce, and then select the waveform's **peak voltage**, the **wave frequency**, and the board's **sample frequency**.

4. When you've completed the setup, click **OK**.
5. Select **Start!** to begin outputting a continuous waveform. *Note that the name changes to **Stop!** until the operation completes. You can choose **Stop!** to halt the operation at any time.*
6. Select **ViewOutput!** to display the waveform on your screen. *Figure 7 shows a typical sine wave.*

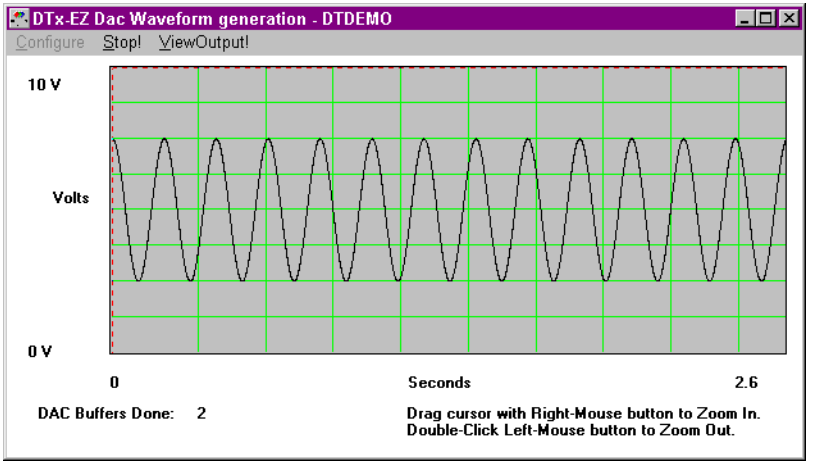


Figure 7: Displaying a Waveform

You can zoom in on a portion of the data by right-clicking on the plot and then dragging the red dotted selection bars.
Double-click the left mouse button to zoom out again.

Continuous A/D Example

This example (contdisp.vbp or contdisp.mdp) continuously samples a single analog input channel to memory and plots the data on screen using pre-defined buffer and frequency settings. You could use this example to detect data trends by immediately viewing the effect of changing stimulus.

Click **Start** to begin the acquisition; click **Stop** to end it. Figure 8 shows a typical acquisition.

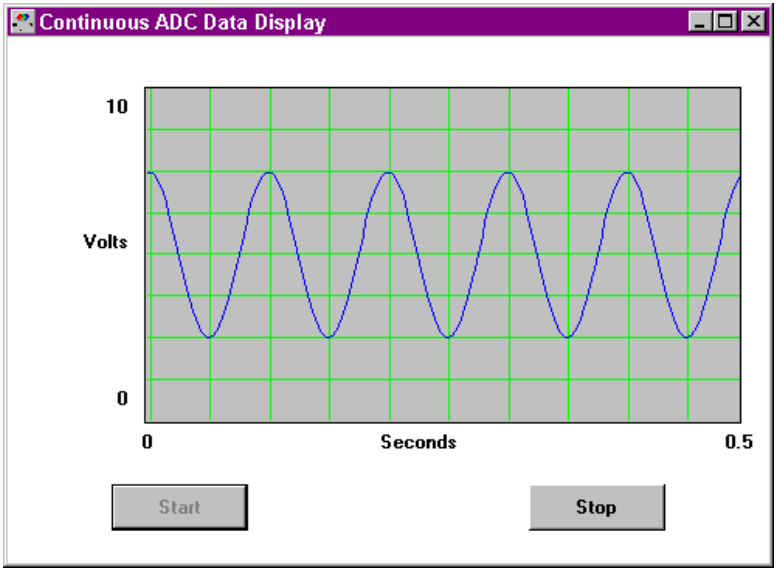


Figure 8: Continuous A/D Display

The Y-axis setting (volts) corresponds to the minimum and maximum voltage settings for the selected board. The X-axis setting (seconds) is determined by the data buffer size and selected sampling frequency. (You can modify these properties within the example's form_load() event subroutine.)

Single-Value Example

This example (sv.vbp or sv.mdp) acquires a single sample from a single analog input channel and outputs a single value on the DAC you specify. You can use this example to check for correct configuration, to monitor slowly-changing inputs, or to provide a constant or slowly-varying voltage output.

When you run it, the example prompts you for the board's name. After the board is located, a display allows you to monitor inputs and generate outputs. A single value is continuously input and displayed in a text box and on the scroll bar, as shown in Figure 9.

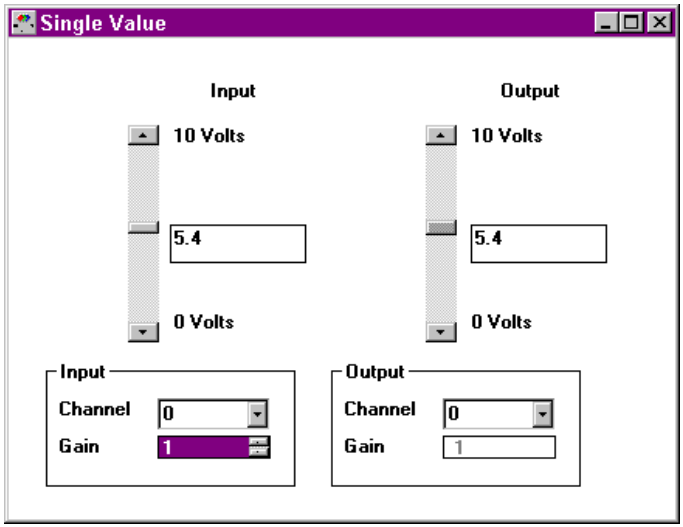


Figure 9: Monitoring Inputs and Outputs for a Single-Value Operation

A single value can also be output continuously. You can change the output voltage setting by entering a new value in the text box or by adjusting the scroll bar.

Note: The actual read and write rate is set (to 100 ms) by the timer control. The clock timer control runs from the system clock.

About-Trigger Example

This example (abouttrig.vbp or abouttrig.mdp) samples a single analog input channel to memory and plots the data on screen using pre-defined buffer and frequency settings until the main trigger event. After the trigger, the example samples the channel for one second and then stops. You could use this example to collect data before and after a specific event (trigger) occurs. In this example, the marker is used to show the first point collected after the trigger event, so you can see the triggering point.

This example also demonstrates using the stripchart mode to plot entire buffers of data rather than single points, one at a time.

Click **Start** to begin the acquisition; click **Stop** to end it. Figure 10 shows a typical acquisition.

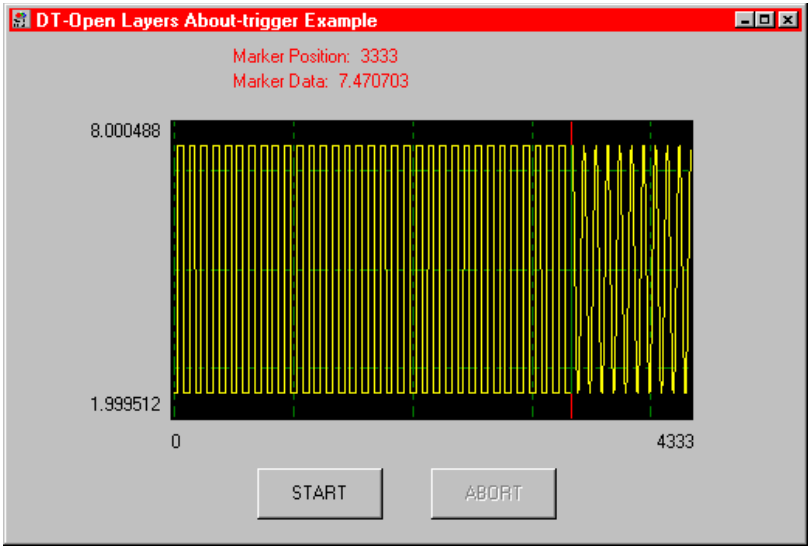


Figure 10: About-Trigger A/D Display

The y-axis setting (volts) corresponds to the minimum and maximum voltage settings for the selected board. The x-axis setting (number of samples) is determined by the data buffer size and selected sampling frequency. (You can modify these properties within the example's form_load() event subroutine.)

Digital I/O Example

This example (dio.vbp or dio.mdp) demonstrates the use of a single value operation with DIN and DOUT subsystems. You could use this example to interface with sensors and control devices that use digital signals.

After selecting the board, the screen shown in Figure 11 appears.

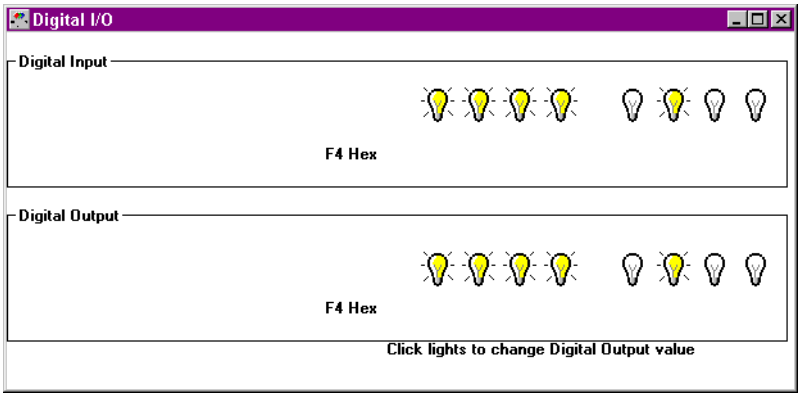


Figure 11: Monitoring Digital I/O Operations

The “light bulbs” represent the digital input and output data. A timer control reads the digital input data at regular intervals. The light bulbs turn “on” and “off” to indicate the value read from the DIN subsystem.

You can change the digital output value by clicking the light bulbs in the bottom half of the display. As you turn them on and off, the new values are written to the DOUT subsystem.

Note: The actual read and write rate is set (to 100 ms) by the timer control. The timer control runs from the system clock.

DDE Server and Client Examples

The DDE server and client examples demonstrate the use of Windows Dynamic Data Exchange (DDE) between applications. In Visual Basic, these examples (server.vbp and client.vbp) reside in the same directory; in Visual C++, these examples (server.mdp and client.mdp) reside in their own directories. In either platform, the two examples are designed to be used together. These examples allow you to send acquired voltage values to other applications for analysis and display and to move voltage values calculated in another application to DTx-EZ for conversion and output. You could use this example for report generation and data analysis or data generation using any Windows spreadsheet, word processing, or analysis package.

For more information on DDE operation, refer to your development environment's programming guide.

Open the Server and Client examples to display the forms shown in Figure 12.

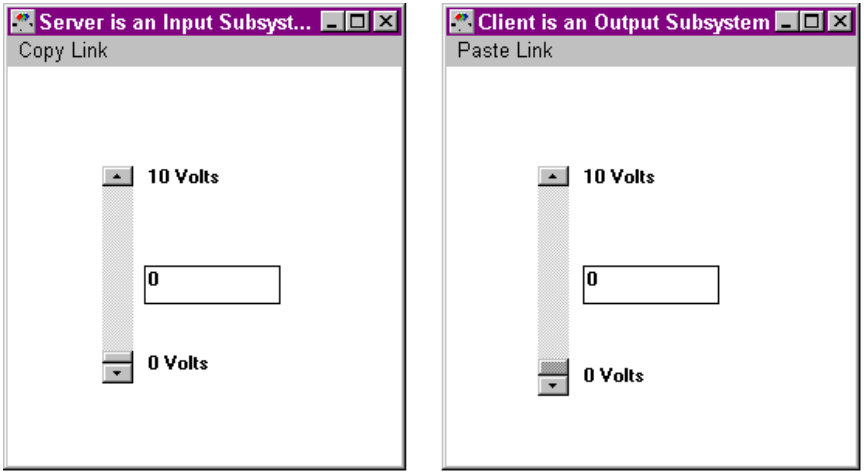


Figure 12: DDE Forms

These forms allow you to share data between the two examples. The Server represents the A/D subsystem and the Client represents the D/A subsystem. The Server continuously updates the input data values in the scroll bar and text box.

Follow these steps:

1. Click **Copy Link** on the Server form to copy the data link to the clipboard.
2. Click **Paste Link** on the Client form to continuously output data from the clipboard and to display it in the Client form's scroll bar and text box.
   Note that the Paste Link command has toggled to ***Close DDE Link****.*
3. Click **Close DDE Link** to stop the data exchange into the D/A subsystem.

Waveform Generator Example

This example (wavegen.vbp or wavegen.mdp) generates a square wave from counter/timer 0. Set the frequency by entering the value (in hertz) in the text box at runtime, as shown in Figure 13.

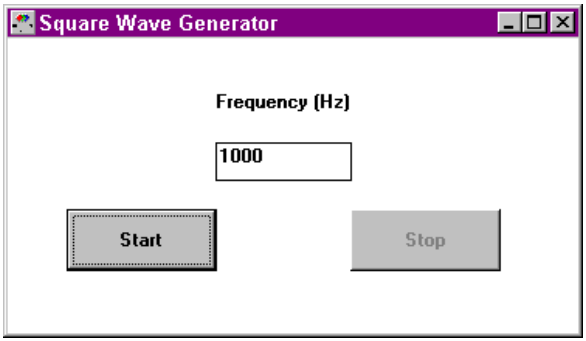


Figure 13: Setting the Waveform Frequency

Continuous FFT Example

This example (contfft.vbp or contfft.mdp) acquires data from a single analog input channel, performs frequency analysis on the data using an FFT (Fast Fourier Transform), and plots the result of the analysis. You could use this example for vibration analysis, to calculate transfer functions, or to monitor the frequency content of an audio signal.

Click **Start** to begin the FFT; click **Stop** to end it. A typical FFT display is shown in Figure 14.

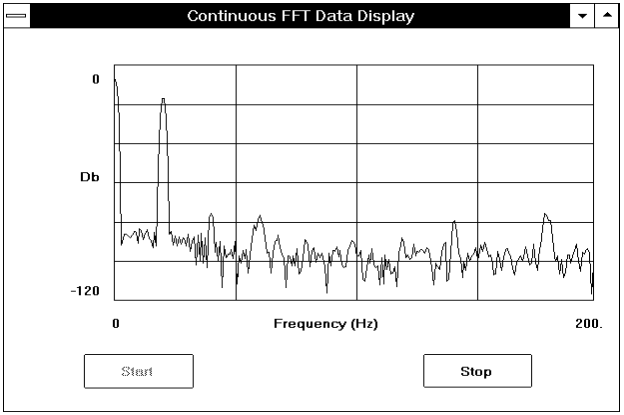


Figure 14: FFT Display

ChartIt Example

This example (chartit.vbp or chartit.mdp) acquires data using the **GetSingleValue** method and plots the data using the plotting control's **SinglePoint** property in stripchart mode.

Click **Start** to begin sampling. A timer control samples and plots each data point every 500 ms. Click **Stop** to halt sampling. A typical stripchart display is shown in Figure 15.

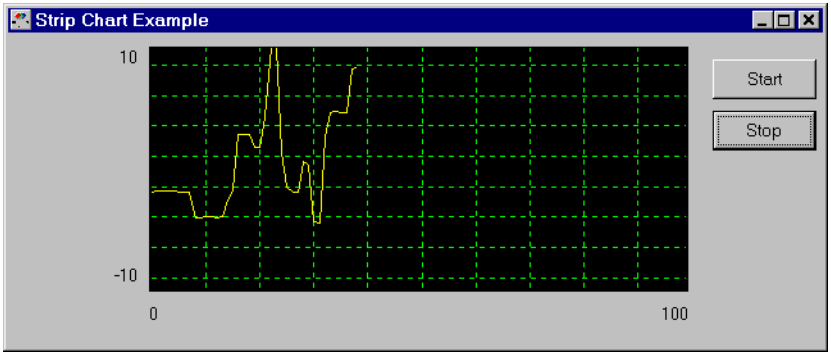


Figure 15: Stripchart Display

Scope Example

This example generates and displays data from one to four analog input channels. It includes typical oscilloscope functions such as single sweep or continuous scan, horizontal and vertical offsets, and adjustable time and amplitude settings. The example also operates as a single-channel spectrum analyzer with a number of popular windowing selections.

You may apply gains and filters (if available) and even invert Channel 2 data and add it to Channel 1 data. Both manual and auto triggering as well as external triggering are possible. By clicking the mouse on the plot display, an exact voltage reading for each channel and the time of acquisition is displayed.

For further analysis, you can apply various windowing formulas as well as FFTs to a single channel of data. Both of these are performed at a user-selected acquisition rate.

Starting the example produces a screen similar to that shown in Figure 16.

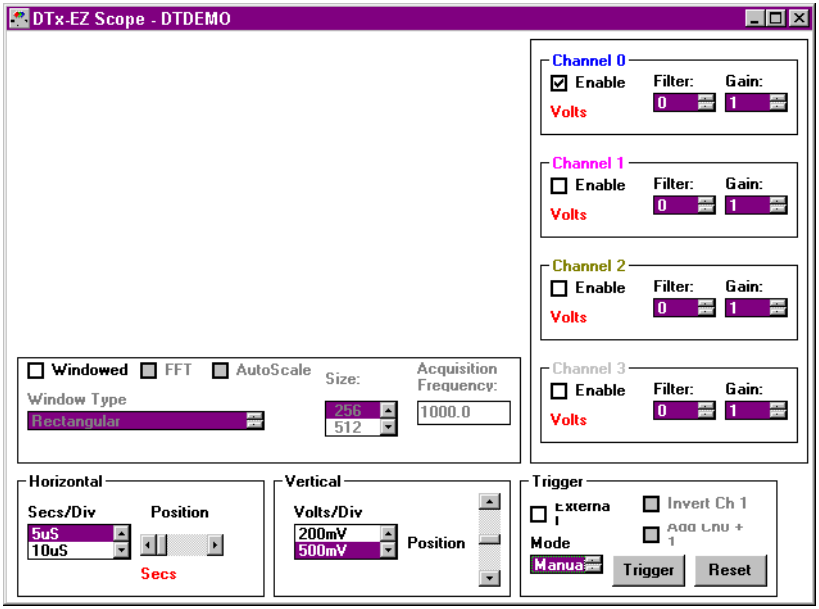


Figure 16: Sample Scope Screen

Property, Method, Function, and Subroutine Summary

Introduction. 46
Data Acquisition Custom Control 47
Data Management DLLs 64
DT Plot Custom Control 69

Introduction

This chapter summarizes the properties, methods, functions, and subroutines provided by the Data Acquisition and the DT Plot Custom Controls in DTx-EZ.

Data Acquisition Custom Control

The Data Acquisition Custom Control provides the following categories of data acquisition tools:

- Information properties and methods (page 47);
- Initialization properties (page 57);
- Configuration properties and functions (page 58); and
- Operation properties, methods, functions, and subroutines (page 61).

The following subsections briefly describe these tools.

Note: For specific information about each of these tools, refer to the DTx-EZ online help. See page 13 for information on launching the online help file.

Information Properties and Methods

To determine the capabilities of your installed boards, subsystems on each board, and software, use the information properties and methods listed in Table 4.

Table 4: Information Properties and Methods

Query About	Properties and Methods	Description
Boards and Devices	BoardList Property	Lists all currently-installed DT-Open Layers data acquisition boards (devices).
	numBoards Property	Returns the number of DT-Open Layers boards currently installed in the system.
	EnumBoards Method	Invokes the board enumeration sequence.
	DeviceName Property	Gets the full name of the specified device (this name is set by the driver as part of the installation procedure).
	hDev Property	Returns the handle of the current subsystem's device.
Subsystems	numSubSystems Property	Returns the number of available subsystems for the selected DT-Open Layers board.
	SubSystemList Property	Lists the subsystems available for the selected DT-Open Layers board.
	EnumSS Method	Lists the names, types, and element number for each subsystem supported by the specified device.
	GetDevCaps Method	Returns the number of elements for a specified subsystem type on a specified device.

Table 4: Information Properties and Methods (cont.)

Query About	Properties and Methods	Description
Subsystems (cont.)	GetSSCaps Method	Returns whether a specified subsystem capability is supported and/or the number of capabilities supported. Refer to Table 5 for a list of possible capabilities and return values.
	GetSSCapsEx Method	Gets information about extended subsystem capabilities including the minimum and maximum throughput, retrigger frequency, clock divider value, and base clock frequency.
	EnumSSCaps Method	Lists the possible settings for specified subsystem capabilities including filters, ranges, gains, and resolution.
	numFilters Property	Returns the number of available filter settings for a subsystem.
	FilterValues Property	Lists filters available to the selected subsystem.
	numGains Property	Returns the number of available gain settings for a subsystem.
	GainValues Property	Lists the subsystem's available gain values.
	numResolutions Property	Returns the number of available resolution settings for a subsystem.
	ResolutionValues Property	Lists a subsystem's available resolution values.
	numRanges Property	Returns the number of available range settings for a subsystem.
	MaxRangeValues Property	Lists a subsystem's maximum voltage range values.

Table 4: Information Properties and Methods (cont.)

Query About	Properties and Methods	Description
Subsystems (cont.)	MinRangeValues Property	Lists a subsystem's minimum voltage range values.
	hDass Property	Returns the handle of the current subsystem.
Software	LastError Property	Retrieves the last known DT-Open Layers error generated by the DTAcq32 Control.
	LastErrorDescription Property	Retrieves a string representation of the last known DT-Open Layers error generated by the DTAcq32 Control.

Table 5 lists the subsystem capabilities that you can query using the **GetSSCaps** method. Note that capabilities may be added as new boards are developed; for the most recent set of capabilities, refer to the DTx-EZ online help.

Table 5: Capabilities to Query with the GetSSCaps Method

Query About	Capability	Method Returns
Data Flow Mode	OLSSC_SUP_SINGLEVALUE	Non-zero if subsystem supports single value operations.
	OLSSC_SUP_CONTINUOUS	Non-zero if subsystem supports continuous (post-trigger) operations.
	OLSSC_SUP_CONTINUOUS_PRETRIG	Non-zero if subsystem supports continuous pre-trigger operations.
	OLSSC_SUP_CONTINUOUS_ABOUTTRIG	Non-zero if subsystem supports continuous about-trigger (both pre- and post-trigger) operations.

Table 5: Capabilities to Query with the GetSSCaps Method (cont.)

Query About	Capability	Method Returns
Simultaneous Operations	OLSSC_SUP_ SIMULTANEOUS_START	Non-zero if subsystem can be started simultaneously with another subsystem on the device.
Pausing Operations	OLSSC_SUP_PAUSE	Non-zero if subsystem supports pausing during continuous operation.
Windows Messaging	OLSSC_SUP_POSTMESSAGE	Non-zero if subsystem supports asynchronous operations.
Buffering	OLSSC_SUP_BUFFERING	Non-zero if subsystem supports buffering.
	OLSSC_SUP_WRPSINGLE	Non-zero if subsystem supports single buffer wrap mode.
	OLSSC_SUP_WRPMULTIPLE	Non-zero if subsystem supports multiple buffer wrap mode.
	OLSSC_SUP_INPROCESS_ FLUSH	Non-zero if subsystem supports copying a buffer on subsystem's inprocess queue.
DMA	OLSSC_NUMDMACHANS	Number of DMA channels supported.
	OLSSC_SUP_GAPFREE_ NODMA	Non-zero if subsystem supports gap-free continuous operation with no DMA.
DMA (cont.)	OLSSC_SUP_GAPFREE_ SINGLEDMA	Non-zero if subsystem supports gap-free continuous operation with a single DMA channel.
	OLSSC_SUP_GAPFREE_ DUALDMA	Non-zero if subsystem supports gap-free continuous operation with two DMA channels.

Table 5: Capabilities to Query with the GetSSCaps Method (cont.)

Query About	Capability	Method Returns
Triggered Scan Mode	OLSSC_SUP_TRIGSCAN	Non-zero if subsystem supports triggered scans.
	OLSSC_MAXMULTISCAN	Maximum number of scans per trigger or retrigger supported by the subsystem.
	OLSSC_SUP_RETRIGGER_SCAN_PER_TRIGGER	Non-zero if subsystem supports scan-per-trigger triggered scan mode (retrigger is the same as the initial trigger source).
	OLSSC_SUP_RETRIGGER_INTERNAL	Non-zero if subsystem supports internal retriggered scan mode. (retrigger source is on the board; initial trigger is any available trigger source).
	OLSSC_SUP_RETRIGGER_EXTRA	Non-zero if subsystem supports retrigger-extra triggered scan mode (retrigger can be any supported trigger source; initial trigger combinations may be limited by the driver).
Channel-Gain List	OLSSC_CGLDEPTH	Number of entries in channel-gain list.
	OLSSC_SUP_RANDOM_CGL	Non-zero if subsystem supports random channel-gain list setup.
	OLSSC_SUP_SEQUENTIAL_CGL	Non-zero if subsystem supports sequential channel-gain list setup.

Table 5: Capabilities to Query with the GetSSCaps Method (cont.)

Query About	Capability	Method Returns
Channel-Gain List (cont.)	OLSSC_SUP_ ZEROSEQUENTIAL_CGL	Non-zero if subsystem supports sequential channel-gain list setup starting with channel zero.
	OLSSC_SUP_ SIMULTANEOUS_SH	Non-zero if subsystem supports simultaneous sample-and-hold operations.The channel-gain list must be set up with both a sample channel and a hold channel.
	OLSSC_SUP_CHANNELLIST_ INHIBIT	Non-zero if subsystem supports channel-gain list entry inhibition.
Gain	OLSSC_SUP_PROGRAMGAIN	Non-zero if subsystem supports programmable gain.
	OLSSC_NUMGAINS	Number of gain selections.
Synchronous Digital I/O	OLSSC_SUP_ SYNCHRONOUS_DIGITALIO	Non-zero if subsystem supports synchronous digital output operations.
	OLSSC_MAXDIGITALIOLIST_ VALUE	Maximum value for synchronous digital output channel list entry.
I/O Channels	OLSSC_NUMCHANNELS	Number of I/O channels.
Channel Type	OLSSC_SUP_SINGLEENDED	Non-zero if subsystem supports single-ended inputs.
	OLSSC_MAXSECHANS	Number of single-ended channels.
	OLSSC_SUP_DIFFERENTIAL	Non-zero if subsystem supports differential inputs.
	OLSSC_MAXDICHANS	Number of differential channels.
Filters	OLSSC_SUP_ FILTERPERCHAN	Non-zero if subsystem supports filtering per channel.
	OLSSC_NUMFILTERS	Number of filter selections.

Table 5: Capabilities to Query with the GetSSCaps Method (cont.)

Query About	Capability	Method Returns
Ranges	OLSSC_NUMRANGES	Number of range selections.
	OLSSC_SUP_ RANGEPERCHANNEL	Non-zero if subsystem supports different range settings for each channel.
Resolution	OLSSC_SUP_ SWRESOLUTION	Non-zero if subsystem supports software-programmable resolution.
	OLSSC_NUMRESOLUTIONS	Number of different resolutions that you can program for the subsystem.
Data Encoding	OLSSC_SUP_BINARY	Non-zero if subsystem supports binary encoding.
	OLSSC_SUP_2SCOMP	Non-zero if subsystem supports twos complement encoding.
Triggers	OLSSC_SUP_SOFTTRIG	Non-zero if subsystem supports internal software trigger.
	OLSSC_SUP_EXTERNTRIG	Non-zero if subsystem supports external digital (TTL) trigger.
	OLSSC_SUP_ THRESHTRIGPOS	Non-zero if subsystem supports positive analog threshold trigger.
	OLSSC_SUP_ THRESHTRIGNEG	Non-zero if subsystem supports negative analog threshold trigger.
	OLSSC_SUP_ ANALOGEVENTTRIG	Non-zero if subsystem supports analog event trigger.
	OLSSC_SUP_ DIGITALEVENTTRIG	Non-zero if subsystem supports digital event trigger.
	OLSSC_SUP_ TIMEREVENTTRIG	Non-zero if subsystem supports timer event trigger.
	OLSSC_ NUMEXTRATRIGGERS	Number of extra trigger sources supported.

Table 5: Capabilities to Query with the GetSSCaps Method (cont.)

Query About	Capability	Method Returns
Clocks	OLSSC_SUP_INTCLOCK	Non-zero if subsystem supports internal clock.
	OLSSC_SUP_EXTCLOCK	Non-zero if subsystem supports external clock.
	OLSSC_NUMEXTRACLOCKS	Number of extra clock sources.
Counter/Timer Modes	OLSSC_SUP_CASCADING	Non-zero if subsystem supports cascading.
	OLSSC_SUP_CTMODE_ COUNT	Non-zero if subsystem supports event counting mode.
	OLSSC_SUP_CTMODE_RATE	Non-zero if subsystem supports rate generation (continuous pulse output) mode.
	OLSSC_SUP_CTMODE_ ONESHOT	Non-zero if subsystem supports (single) one-shot mode.
	OLSSC_SUP_CTMODE_ ONESHOT_RPT	Non-zero if subsystem supports repetitive one-shot mode.
Counter/Timer Pulse Output Types	OLSSC_SUP_PLS_HIGH2LOW	Non-zero if subsystem supports high-to-low output pulses.
	OLSSC_SUP_PLS_LOW2HIGH	Non-zero if subsystem supports low-to-high output pulses

Table 5: Capabilities to Query with the GetSSCaps Method (cont.)

Query About	Capability	Method Returns
Counter/Timer Gates	OLSSC_SUP_GATE_NONE	Non-zero if subsystem supports an internal (software) gate type.
	OLSSC_SUP_GATE_HIGH_ LEVEL	Non-zero if subsystem supports high-level gate type.
	OLSSC_SUP_GATE_LOW_ LEVEL	Non-zero if subsystem supports low-level gate type.
	OLSSC_SUP_GATE_HIGH_ EDGE	Non-zero if subsystem supports high-edge gate type.
	OLSSC_SUP_GATE_LOW_ EDGE	Non-zero if subsystem supports low-edge gate type.
	OLSSC_SUP_GATE_LEVEL	Non-zero if subsystem supports level change gate type.
	OLSSC_SUP_GATE_HIGH_ LEVEL_DEBOUNCE	Non-zero if subsystem supports high-level gate type with input debounce.
	OLSSC_SUP_GATE_LOW_ LEVEL_DEBOUNCE	Non-zero if subsystem supports low-level gate type with input debounce.
	OLSSC_SUP_GATE_HIGH_ EDGE_DEBOUNCE	Non-zero if subsystem supports high-edge gate type with input debounce.
	OLSSC_SUP_GATE_LOW_ EDGE_DEBOUNCE	Non-zero if subsystem supports low-edge gate type with input debounce.
	OLSSC_SUP_GATE_LEVEL_ DEBOUNCE	Non-zero if subsystem supports level change gate type with input debounce.
Interrupt	OLSSC_SUP_INTERRUPT	Non-zero if subsystem supports interrupt-driven I/O.

Table 5: Capabilities to Query with the GetSSCaps Method (cont.)

Query About	Capability	Method Returns
FIFOs	OLSSC_SUP_FIFO	Non-zero if subsystem has a FIFO in the data path.
Processors	OLSSC_SUP_PROCESSOR	Non-zero if subsystem has a processor on board.
Software Calibration	OLSSC_SUP_SWCAL	Non-zero if subsystem supports software calibration.

Initialization Properties

Once you have identified the available devices, use the initialization properties described in Table 6.

Table 6: Initialization Properties

Property	Description
Board Property	Provides the means for the software to associate specific requests with a particular board; it must be called before any other property. This property loads a specified board's software support. Specify the alias name assigned to the board upon its installation.
SubSystem Property	Provides the means for the software to associate specific requests with a particular subsystem on a board; it must be called after the **Board** property and before any other tool.
SubSysElement Property	Sets and returns the subsystem's element number. May be used in conjunction with the **SubSysType** property as a replacement for the **SubSystem** property.
SubSysType Property	Sets and returns the subsystem's type. May be used in conjunction with the **SubSysElement** property as a replacement for the **SubSystem** property.

Configuration Properties and Functions

Once you have initialized a board and subsystem and determined what its capabilities are, set or return the value of the subsystem's parameters using the configuration properties and functions listed in Table 7.

Note that *italic text* indicates a the name of an alternate function call.

Table 7: Configuration Properties and Functions

Feature	Properties and Functions	Description
Data Flow Mode	DataFlow Property	Sets and returns the data flow mode.
Buffer Wrap Mode	WrapMode Property	Sets and returns the buffer processing wrap mode.
DMA	DmaUsage Property	Sets and returns the number of DMA channels to be used.
Triggered Scans	TriggeredScan Property	Enables or disables triggered scan mode.
	MultiscanCount Property	Sets and returns the number of times to scan per trigger/retrigger.
	RetriggerMode Property	Sets and returns the retrigger mode.
	RetriggerFreq Property	Sets and returns the frequency of the internal retrigger when using internal retrigger mode.

TEquipment USA .NET
An Interworld Highway, LLC Company

Table 7: Configuration Properties and Functions (cont.)

Feature	Properties and Functions	Description
Channel-Gain List	ListSize Property	Sets and returns the size of the channel-gain list.
	ChannelList Property	Sets and returns the channel number of a channel-gain list entry.
	GainList Property	Sets a gain value for a channel-gain list entry.
	InhibitList Property	Enables/disables channel entry inhibition for a channel-gain list entry.
	DIOList Property	Sets and returns the digital value to output for the channel-gain list entry.
Synchronous Digital I/O	SyncDIOUsage Property	Enables/disables synchronous digital I/O operations.
Channel Type	ChannelType Property	Sets and returns the channel configuration type of a channel.
Filters	FilterList Property	Sets and returns the analog filter that may be applied to each input or output channel.
Ranges	Range Property	Sets and returns the voltage range for a subsystem.
	Get Channel Range Function olDaGetChannelRange	Gets the voltage range for a channel.
	MaxRange Property	Returns the maximum voltage value of the current range setting.
	MinRange Property	Returns the minimum voltage value of the current range setting.

Table 7: Configuration Properties and Functions (cont.)

Feature	Properties and Functions	Description
Resolution	Resolution Property	Sets and returns the number of bits of resolution.
Data Encoding	Encoding Property	Sets and returns the data encoding type.
Triggers	Trigger Property	Sets and returns the post-trigger source.
	PreTrigger Property	Sets and returns the pre-trigger source.
	ReTrigger Property	Sets and returns the retrigger source for retrigger-extra retrigger mode.
Clocks	ClockSource Property	Sets and returns the clock source.
	Frequency Property	Sets and returns the frequency of the internal clock or a counter/timer's output frequency.
	ClockDivider Property	Sets and returns the divider value applied to the external clock.
Counter/Timers	CTMode Property	Sets and returns the counter/timer mode.
	CascadeMode Property	Sets and returns the counter/timer cascade mode.
	GateType Property	Sets and returns the gate type for the counter/timer mode.
	PulseType Property	Sets and returns the pulse type for the counter/timer mode.
	PulseWidth Property	Sets and returns the pulse output width for the counter/timer mode.

Operation Properties, Methods, Functions, and Subroutines

Once you have set the parameters of a subsystem, use the operation properties, methods, functions, and subroutines listed in Table 8. Note that *italic text* indicates a the name of an alternate function call.

Table 8: Operation Properties, Methods, Functions, and Subroutines

Operation	Properties, Methods, Functions, and Subroutines	Description
Single-Value Operations	GetSingleValue Method	Reads a single input value from the specified subsystem channel.
	PutSingleValue Method	Writes a single output value to the specified subsystem channel.
All Other Operations	Config Method	After setting up a specified subsystem using the configuration tools, configures the subsystem with new parameter values.
	Start Method	Starts the operation for which the subsystem has been configured.
	Pause Method	Pauses a continuous operation on the subsystem.
	Continue Method	Continues the previously paused operation on the subsystem.
	Stop Method	Stops the operation and returns the subsystem to the ready state.
	Abort Method	Stops the subsystem's operation immediately.
	Reset Method	Causes the operation to terminate immediately, and reinitializes the subsystem.

Table 8: Operation Properties, Methods, Functions, and Subroutines (cont.)

Operation	Properties, Methods, Functions, and Subroutines	Description
Buffer Operations	Flush Method	Transfers all data buffers held by the subsystem to the done queue.
	Queue Property	Adds buffers to the ready queue and retrieves buffers from the done queue.
	QueueSize Property	Gets the size of the specified queue (ready, done or inprocess) for a specified subsystem. The size indicates the number of buffers on the specified queue.
Counter/Timer Operations	CTReadEvents Method	Gets the number of events that have been counted since the subsystem was started with the **Start** method.
	MeasureFrequency Method	Measures the frequency of the input clock source for the selected counter/timer.
Power Operations	PowerOn Method	Powers on a USB module and restores the configuration of the module at the time that it was last powered down.
	PowerOff Method	Stores the configuration of the USB module and powers down the module.
Errors	ClearError Method	Clears the **LastErrNum** property.
	LastErrNum Property	Retrieves the last known DT-Open Layers error generated by the DTAcq32 Control.
	LastErrDescription Property	Retrieves a string representation of the last known DT-Open Layers error generated by the DTAcq32 Control.

Table 8: Operation Properties, Methods, Functions, and Subroutines (cont.)

Operation	Properties, Methods, Functions, and Subroutines	Description
Simultaneous Operations	GetSimultaneousStartList Function olDaGetSSList	Creates a simultaneous start list and returns a handle to it.
	PutSubSysOnSSList Subroutine olDaPutDassToSSList	Puts the specified subsystem on the simultaneous start list.
	SimultaneousPreStart Subroutine olDaSimultaneousPreStart	Simultaneously prestarts (performs setup operations on) all subsystems on the specified simultaneous start list.
	SimultaneousStart Subroutine olDaSimultaneousStart	Simultaneously starts all subsystems on the specified simultaneous start list.
	ReleaseSimultaneousStartList olDaReleaseSSList	Releases all subsystems from the simultaneous start list and removes the list itself.

Data Management DLLs

In addition to the Data Acquisition Custom Control, DTx-EZ offers the following data management tools:

- Buffer management functions and subroutines (page 64) and
- Conversion functions and subroutines (page 67).

Buffer Management Functions and Subroutines

The buffer management functions and subroutines form one of the basic elements of the DT-Open Layers architecture. They "glue" the various layers together. The fundamental data object in DTx-EZ is a buffer. All functions that create, manipulate, and delete buffers are encapsulated in the data management portion of DTx-EZ.

The buffer management functions and subroutines, listed in Table 9, are intended for use by both application and system programmers. They provide a set of object-oriented buffer management facilities. When a buffer object is created, a buffer handle (*hbuf*) is returned. This handle is used in all subsequent buffer manipulation.

Note: The buffer management functions and subroutines, listed in Table 9, are intended for use by both application and system programmers. They provide a set of object-oriented buffer management facilities. When a buffer object is created, a buffer handle (hbuf) is returned. This handle is used in all subsequent buffer manipulation.

Note: Because of the differences in the two compliers, the Visual Basic and Visual C++ libraries exist as separate entities; however, they are nearly identical in functionality. In Table 9, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ name appears in italics following the Visual Basic name.

For specific information about each of these functions and subroutines, refer to the DTx-EZ online help. See page 13 for information on launching the online help file.

Table 9: Buffer Management Functions and Subroutines

Functions and Subroutines	Description
AllocBuffer Function olDmAllocBuffer	Creates a buffer object of a specified number of samples, where each sample is 2 bytes.
CallocBuffer Function olDmCallocBuffer	Creates a buffer object of a specified number of samples of a specified size.
CopyChannelFromBuffer Subroutine*	Copies one selected channel's data from a buffer to the specified array.
CopyLongChannelFromBuffer Subroutine*	Copies one selected channel's data from a buffer to the specified array.
CopySingleChannelFromBuffer Subroutine*	Copies one selected channel's data from a buffer to the specified array.
CopyChannelToBuffer Subroutine*	Copies one selected channel's data from a buffer to the specified array.
CopyLongChannelToBuffer Subroutine*	Copies one selected channel's data from a buffer to the specified array.

* These functions were instituted for Visual Basic users since direct buffer access cannot be achieved; Visual C++ users can access the buffer directly with *olDmGetBufferPtr.*

Table 9: Buffer Management Functions and Subroutines (cont.)

Functions and Subroutines	Description
CopySingleChannelToBuffer Subroutine*	Copies one selected channel's data from a buffer to the specified array.
CopyFromBuffer Subroutine*	Copies data from a buffer to the specified array.
CopyToBuffer Subroutine*	Copies data from an array to the specified buffer.
FreeBuffer Subroutine olDmFreeBuffer	Deletes a buffer object.
olDmGetBufferPtr	Gets a pointer to the buffer data.
GetBufferSize Function olDmGetBufferSize	Gets the physical buffer size (in bytes).
GetDataBits Function olDmGetDataBits	Gets the number of valid data bits.
GetDataWidth Function olDmGetDataWidth	Gets the width of each data sample.
GetErrorString Function olDaGetErrorString olDmGetErrorString olDspGetErrorString	Gets the string corresponding to a data management error code value.
GetMaxSamples Function olDmGetMaxSamples	Gets the physical size of the buffer (in samples).
GetTimeDateStamp Function olDmGetTimeDateStamp	Gets the time and date of the buffer's data.
SetValidSamples Function olDmSetValidSamples	Sets the number of valid samples in the buffer.

Table 9: Buffer Management Functions and Subroutines (cont.)

Functions and Subroutines	Description
GetValidSamples Function olDmGetValidSamples	Gets the number of valid samples.
ReallocBuffer Subroutine olDmReAllocBuffer	Reallocates a buffer object (alloc() interface).
ReCallocBuffer Subroutine olDmReCallocBuffer	Reallocates a buffer object (calloc() interface).

Conversion Functions and Subroutines

The data conversion utilities that are available in DTx-EZ are listed in Table 10.

Note: The conversion utilities, listed in Table 10, are intended for use by both application and system programmers. Because of the differences in the two compliers, the Visual Basic and Visual C++ libraries exist as separate entities; however, they are nearly identical in functionality. In Table 10, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ name appears in italics following the Visual Basic name.

For specific information about each of these functions and subroutines, refer to the DTx-EZ online help. See page 13 for information on launching the online help file.

Table 10: Conversion Utilities

Functions and Subroutines	Description
ValueToVolts Function	Converts a value into a voltage value as a single-precision value.
VoltsToOutput Subroutine olDspVoltsToOutput	Converts input voltage values to an output buffer that is compatible with the current setting of the specified subsystem.
VoltsToValue Function	Converts the specified voltage into units that are appropriate for the specified subsystem.
InputToVolts Subroutine olDspInputToVolts	Converts a subsystem input buffer into the corresponding voltage values.
MagToDB Subroutine olDspMagToDB	Converts the input buffer data into decibels (dB).
RealFFT Subroutine olDspRealFFT	Performs a Fast Fourier Transform (FFT) on the specified data buffer.
Window Subroutine olDspWindow	Converts the input data buffer into floating-point and applies the specified window.

DT Plot Custom Control

The DT Plot Custom Control provides the following categories of plotting control properties:

- Plot appearance (this page),
- Plot pre-display operational parameters (page 70),
- Grids (page 71),
- Markers (page 71),
- x-Axis parameters (page 72),
- y-Axis parameters (page 73), and
- Plotting operation control parameters (page 73).

The following subsections briefly describe these properties.

Note: For specific information about each of these properties, refer to the DTx-EZ online help. See page 13 for information on launching the online help file.

Plot Appearance

The properties outlined in Table 11 allow you to affect the display's basic appearance.

Table 11: Plot Appearance

Property	Description
BackColor	Sets the display's background color (Microsoft standard property)
ForeColor	Sets the plot lines' colors (Microsoft standard property)
Palette Property	Sets the color of each channel's plot line individually.
LineStyle Property	Sets the style of the data plotting lines.
LineWidth Property	Sets the width of the data plotting lines.

Plot Pre-Display Operational Parameters

The properties outlined in Table 12 allow you to define how the plot functions and outputs data.

Table 12: Plot Pre-Display Operational Parameter Properties

Property	Description
StripChartMode Property	Enables/Disables the stripchart mode.
StripChartSize Property	Sets and returns the maximum number of data points to store and that can be displayed per channel when in stripchart mode.
DataType Property	Sets the type of data in the buffer object (unsigned fixed point, singed fixed point, or floating-point).
numChannels Property	Specifies the number of data channels in the buffer object.

Grids

The properties in Table 13 allow you to affect the display's grid appearance.

Table 13: Grid Properties

Property	Description
GridAutoScale Property	Enables/Disables grid autoscale mode.
GridColor Property	Sets and returns the grid color.
GridStyle Property	Sets and returns the grid's line style.
GridXOn Property	Displays/Hides vertical grid lines.
GridXSpacing Property	Sets vertical grid line spacing.
GridXStart Property	Sets the position of the first vertical grid line.
GridYOn Property	Displays/Hides horizontal grid lines.
GridYSpacing Property	Sets horizontal grid line spacing.
GridYStart Property	Sets the position of the first horizontal grid line.

Markers

The properties outlined in Table 14 allow you to affect the display's marker appearance.

Table 14: Marker Properties

Property	Description
MarkerColor Property	Sets the color for the horizontal and vertical markers.
MarkerH1On Property	Displays/Hides the first horizontal marker.

Table 14: Marker Properties (cont.)

Property	Description
MarkerH1Pos Property	Sets the position of the first horizontal marker.
MarkerH2On Property	Displays/Hides the second horizontal marker.
MarkerH2Pos Property	Sets the position of the second horizontal marker.
MarkerV1Data Property	Sets and returns the value of the data point at the location of the first marker.
MarkerV1On Property	Displays/Hides the first vertical marker.
MarkerV1Pos Property	Sets the position of the first vertical marker.
MarkerV2Data Property	Sets and returns the value of the data point at the location of the second marker.
MarkerV2On Property	Displays/Hides the second vertical marker.
MarkerV2Pos Property	Sets the position of the second vertical marker.

x-Axis Parameters

The properties outlined in Table 15 allow you to affect a plot's x-axis.

Table 15: x-Axis Parameter Properties

Property	Description
xAutoScale Property	Enables/Disables setting the **xStart** and **xLength** properties automatically for each new data buffer.
xLength Property	Specifies the amount of data per channel to display.
xScale Property	Sets the x-axis scaling of the input data buffer.
xStart Property	Sets the first data point to be displayed.

y-Axis Parameters

The properties outlined in Table 16 allow you to affect a plot's y-axis.

Table 16: x-Axis Parameter Properties

Property	Description
yAutoScale Property	Enables/Disables setting the **yMin** and **yMax** properties automatically for each new data buffer.
yMax Property	Specifies the upper limit of the plot's y-axis.
yMin Property	Specifies the lower limit of the plot's y-axis.

Plotting Operation Control Parameters

The properties outlined in Table 17 allow you to affect how the DT Plot Custom Control handles and plots data.

Table 17: Plotting Operation Control Parameter Properties

Property	Description
Buffer Property	Plots a buffer of data.
ForceRepaint Property	Enables/Disables the repainting mode.
MouseXPos Property	Returns the x coordinate of the current mouse position.
MouseYPos Property	Returns the y coordinate of the current mouse position.
SinglePoint Property	Plots a single point in stripchart mode.
UpdateMode Property	Enables/Disables the plot update mode.

Programming Flowcharts

Introduction. 76
Single-Value Operations 77
Continuous Buffered Input Operations 79
Continuous Buffered Output Operations 81
Event Counting Operations 83
Frequency Measurement Operations 85
Pulse Output Operations. 87
Plotting Control Operations 89

Introduction

If you are unfamiliar with the capabilities of your board and/or subsystem, query the device as follows:

- To determine the number and types of DT-Open Layers boards and drivers installed, use the **numBoards** and the **BoardList** properties.
- To determine the subsystems supported by the board, use the **EnumSS** or **GetDevCaps** method.
- To determine the capabilities of a subsystem, use the **GetSSCaps** or **GetSSCapsEx** method, specifying one of the capabilities listed in Table 5 on page 50.
- To determine the gains, filters, resolutions, and ranges if more than one is available, use the **EnumSSCaps** method, or these sets of properties:
  - Gains — **numGains** and **GainValues**
  - Filters — **numFilters** and **FilterValues**
  - Resolutions — **numResolutions** and **ResolutionsValues**
  - Ranges — **numRanges**, **MinRangeValues**, and **MaxRangeValues**

Then, follow the flowcharts presented in the remainder of this chapter to perform the desired operation.

Note: Although the flowcharts do not show error checking, it is recommended that you check for errors after using each property, method, function, and subroutine.

Single-Value Operations

The flowchart in Figure 17 provides an overview of the steps required to perform a single-value operation. Some steps represent several substeps; if you are unfamiliar with the functions required to perform a step, refer to the indicated page in Appendix A for more information.

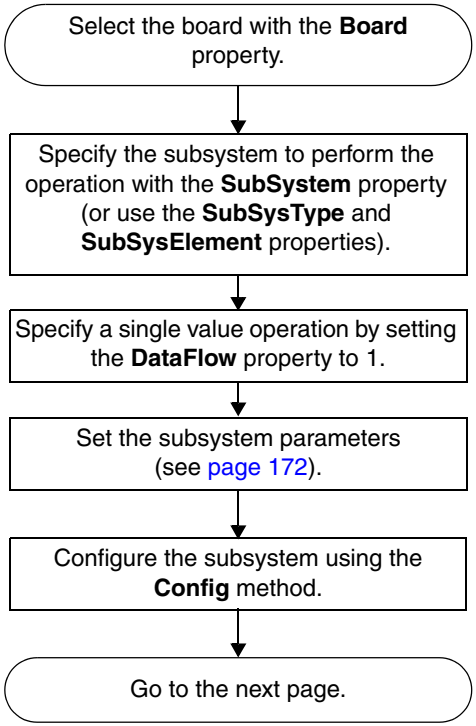

Select the board with the **Board** property.
Specify the subsystem to perform the operation with the **SubSystem** property (or use the **SubSysType** and **SubSysElement** properties).
Specify A/D for an analog input subsystem, D/A for an analog output subsystem, DIN for a digital input subsystem, or DOUT for a digital output subsystem.
Specify a single value operation by setting the **DataFlow** property to 1.
Set the subsystem parameters (see page 172).
Configure the subsystem using the **Config** method.
Go to the next page.

Figure 17: Performing Single-Value Operations

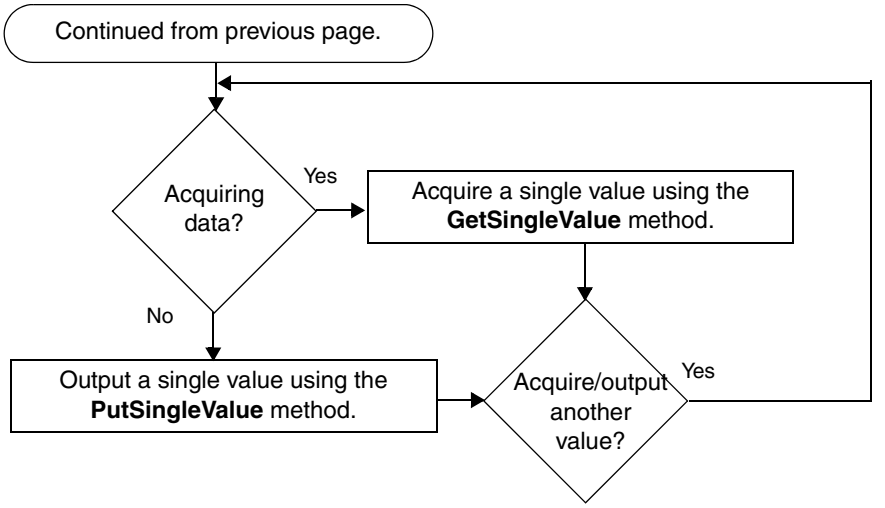
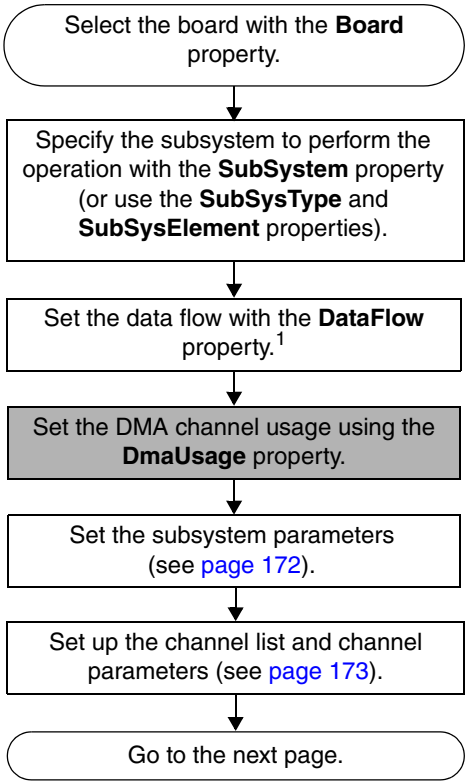


Figure 17: Performing Single-Value Operations (cont.)

Continuous Buffered Input Operations

The flowchart in Figure 18 provides an overview of the steps required to perform a continuous buffered analog input or digital input operation. Many steps represent several substeps; if you are unfamiliar with the functions required to perform a step, refer to the indicated page in Appendix A for more information. Optional steps appear in shaded boxes.


Select the board with the **Board** property.
Specify the subsystem to perform the operation with the **SubSystem** property (or use the **SubSysType** and **SubSysElement** properties).
Set the data flow with the **DataFlow** property.[1]
Set the DMA channel usage using the **DmaUsage** property.
Set the subsystem parameters (see page 172).
Set up the channel list and channel parameters (see page 173).
Go to the next page.

[1] Specify continuous (0) for post-trigger operations, continuous pre-trigger (4) for continuous pre-trigger operations, or continuous about-trigger (5) for continuous about-trigger operations).

Figure 18: Performing a Continuous Buffered Input Operation

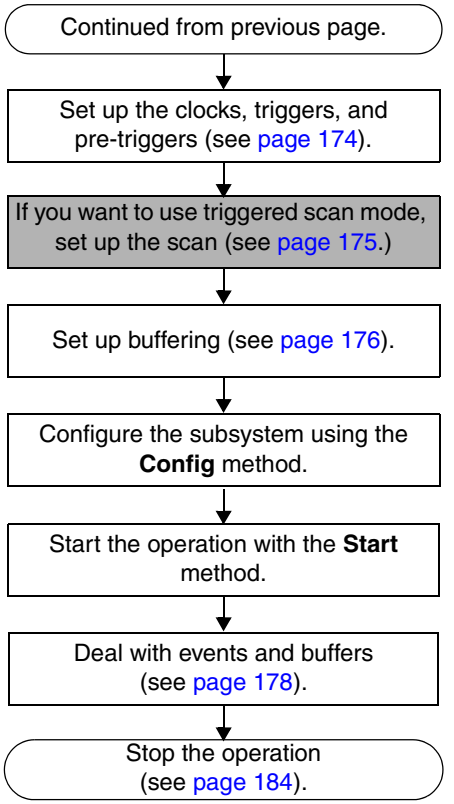

Continued from previous page.
Set up the clocks, triggers, and pre-triggers (see page 174).
If you want to use triggered scan mode, set up the scan (see page 175.)
Set up buffering (see page 176).
Configure the subsystem using the **Config** method.
Start the operation with the **Start** method.
Deal with events and buffers (see page 178).
Stop the operation (see page 184).

Figure 18: Performing a Continuous Buffered Input Operation (cont.)

Continuous Buffered Output Operations

The flowchart in Figure 19 provides an overview of the steps required to perform a continuous buffered analog output or digital output operation. Many steps represent several substeps; if you are unfamiliar with the functions required to perform a step, refer to the indicated page in Appendix A for more information. Optional steps appear in shaded boxes.

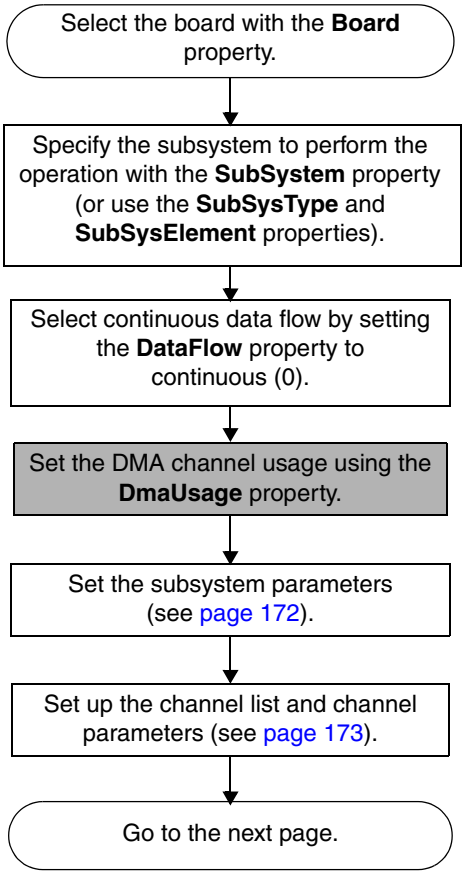

Select the board with the **Board** property.
Specify the subsystem to perform the operation with the **SubSystem** property (or use the **SubSysType** and **SubSysElement** properties).
Select continuous data flow by setting the **DataFlow** property to continuous (0).
Set the DMA channel usage using the **DmaUsage** property.
Set the subsystem parameters (see page 172).
Set up the channel list and channel parameters (see page 173).
Go to the next page.

Figure 19: Performing a Continuous Buffered Output Operation

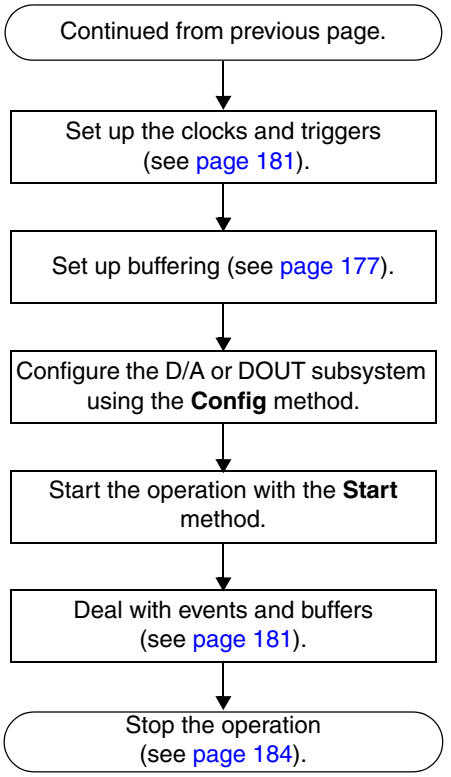

Continued from previous page.
Set up the clocks and triggers (see page 181).
Set up buffering (see page 177).
Configure the D/A or DOUT subsystem using the **Config** method.
Start the operation with the **Start** method.
Deal with events and buffers (see page 181).
Stop the operation (see page 184).

Figure 19: Performing a Continuous Buffered Output Operation (cont.)

Event Counting Operations

The flowchart in Figure 20 provides an overview of the steps required to perform an event counting operation. Many steps represent several substeps; if you are unfamiliar with the functions required to perform a step, refer to the indicated page in Appendix A for more information. Optional steps appear in shaded boxes.

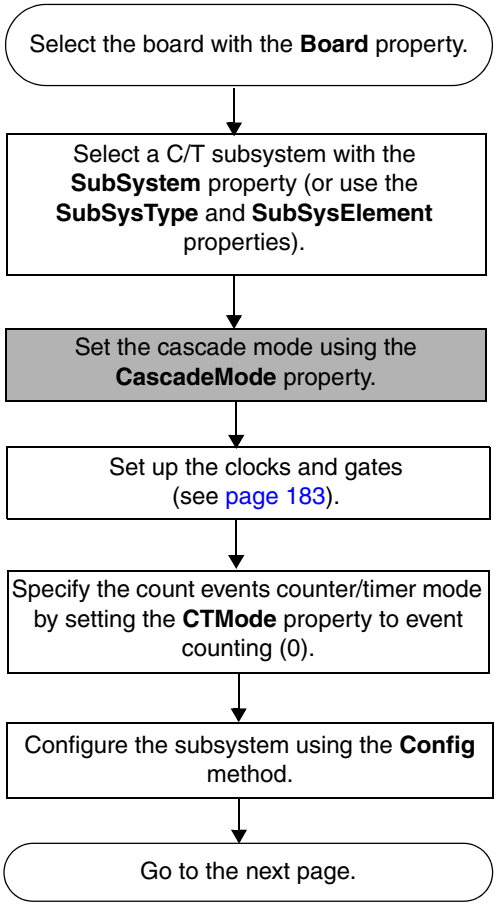

Select the board with the **Board** property.
Select a C/T subsystem with the **SubSystem** property (or use the **SubSysType** and **SubSysElement** properties).
Set the cascade mode using the **CascadeMode** property.
Set up the clocks and gates (see page 183).
Specify the count events counter/timer mode by setting the **CTMode** property to event counting (0).
Configure the subsystem using the **Config** method.
Go to the next page.

Figure 20: Performing an Event Counting Operation

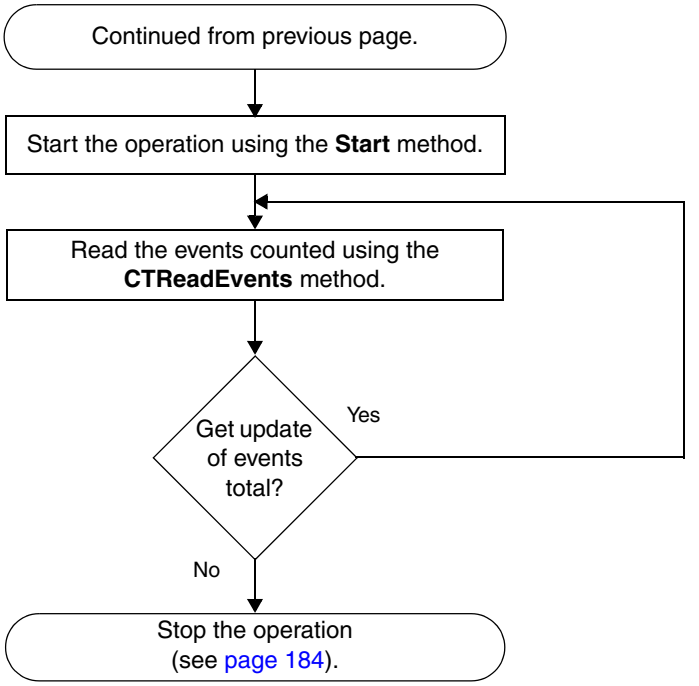

Continued from previous page.
Start the operation using the **Start** method.
Read the events counted using the **CTReadEvents** method.
Get update of events total?
Yes
No
Stop the operation (see page 184).

Figure 20: Performing an Event Counting Operation (cont.)

Frequency Measurement Operations

The flowchart in Figure 21 provides an overview of the steps required to perform a frequency measurement operation. Many steps represent several substeps; if you are unfamiliar with the functions required to perform a step, refer to the indicated page in Appendix A for more information. Optional steps appear in shaded boxes.

Note: If you need more accuracy than the Windows timer provides, refer to page 140.

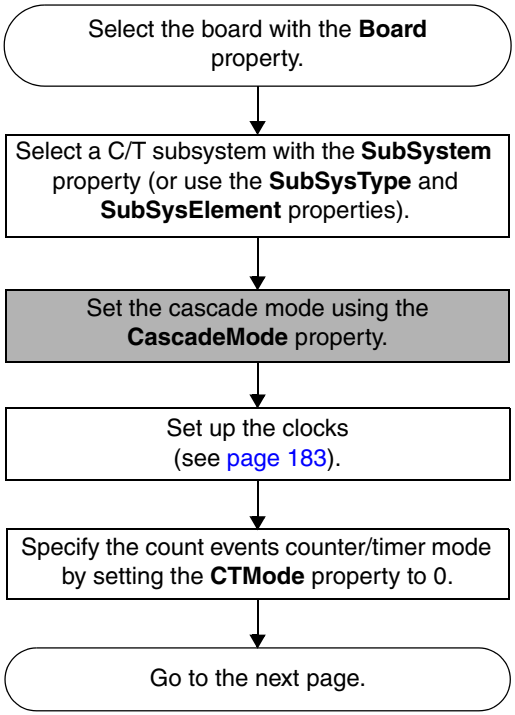

Select the board with the **Board** property.
Select a C/T subsystem with the **SubSystem** property (or use the **SubSysType** and **SubSysElement** properties).
Set the cascade mode using the **CascadeMode** property.
Set up the clocks (see page 183).
Specify the count events counter/timer mode by setting the **CTMode** property to 0.
Go to the next page.

Figure 21: Performing a Frequency Measurement Operation

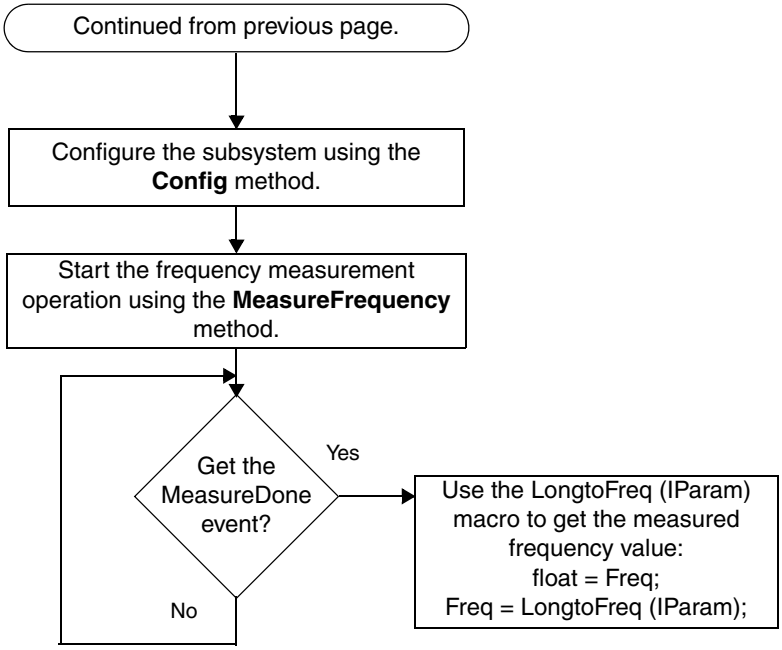
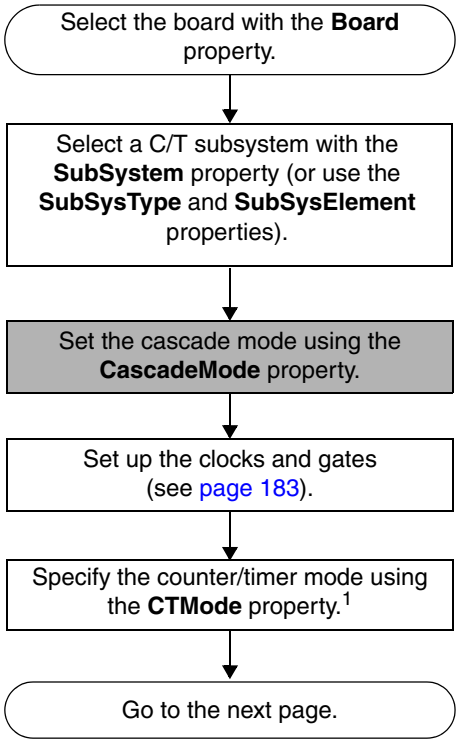

Continued from previous page.
Configure the subsystem using the **Config** method.
Start the frequency measurement operation using the **MeasureFrequency** method.
Get the MeasureDone event?
Yes
No
Use the LongtoFreq (IParam) macro to get the measured frequency value:
float = Freq;
Freq = LongtoFreq (IParam);

Figure 21: Performing a Frequency Measurement Operation (cont.)

Pulse Output Operations

The flowchart in Figure 22 provides an overview of the steps required to perform a pulse output operation, including rate generation, single one-shot, or repetitive one-shot. Many steps represent several substeps; if you are unfamiliar with the functions required to perform a step, refer to the indicated page in Appendix A for more information. Optional steps appear in shaded boxes.


Select the board with the **Board** property.
Select a C/T subsystem with the **SubSystem** property (or use the **SubSysType** and **SubSysElement** properties).
Set the cascade mode using the **CascadeMode** property.
Set up the clocks and gates (see page 183).
Specify the counter/timer mode using the **CTMode** property.[1]
Go to the next page.

[1] Specify 1 for rate generation (continuous pulse output), 2 for single one-shot, or 3 for repetitive one-shot.

Figure 22: Performing a Pulse Output Operation

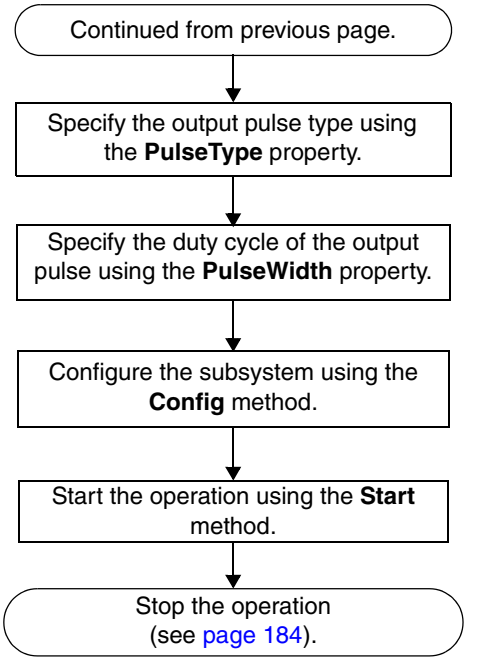

Continued from previous page.
Specify the output pulse type using the **PulseType** property.
Specify the duty cycle of the output pulse using the **PulseWidth** property.
Configure the subsystem using the **Config** method.
Start the operation using the **Start** method.
Stop the operation
(see page 184).

Figure 22: Performing a Pulse Output Operation (cont.)

Plotting Control Operations

The flowchart in Figure 23 provides an overview of the steps required to plot data. Many steps represent several substeps; if you are unfamiliar with the functions required to perform a step, refer to the indicated page in Appendix A for more information. Optional steps appear in shaded boxes.

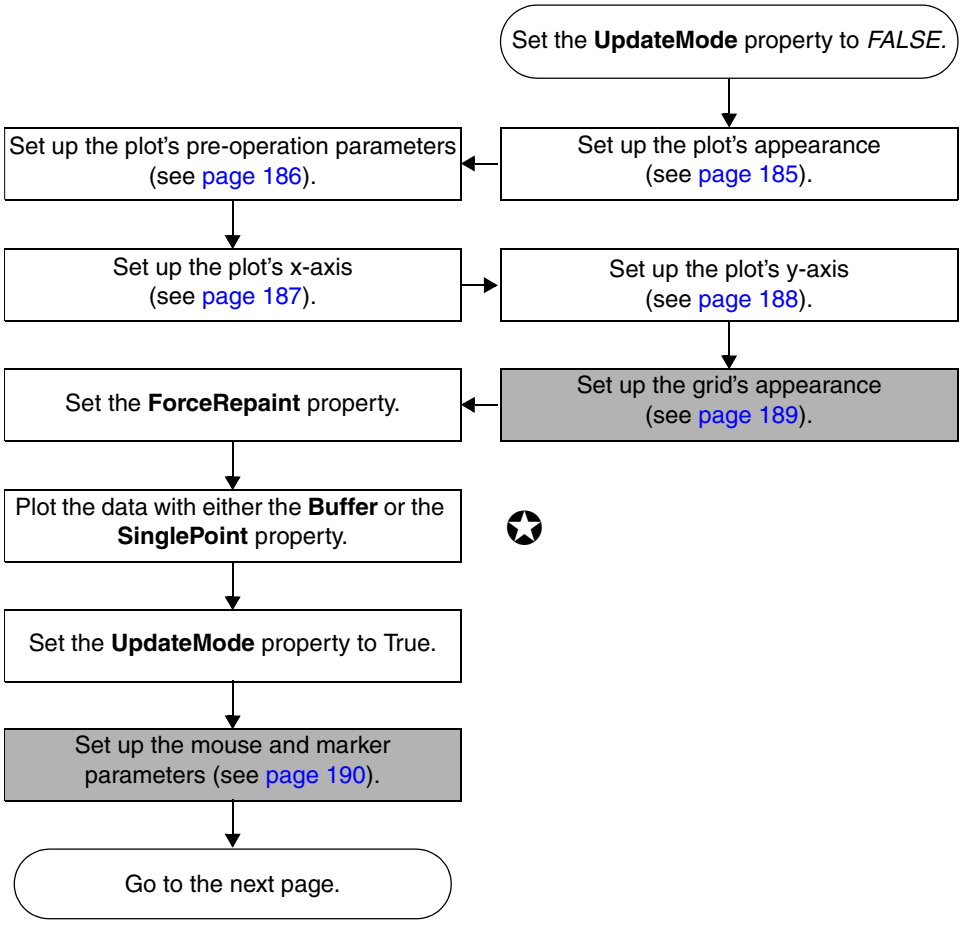

Set the **UpdateMode** property to *FALSE*.
Set up the plot's appearance (see page 185).
Set up the plot's pre-operation parameters (see page 186).
Set up the plot's x-axis (see page 187).
Set up the plot's y-axis (see page 188).
Set up the grid's appearance (see page 189).
Set the **ForceRepaint** property.
Plot the data with either the **Buffer** or the **SinglePoint** property.
Set the **UpdateMode** property to True.
Set up the mouse and marker parameters (see page 190).
Go to the next page.

Figure 23: Plotting Data

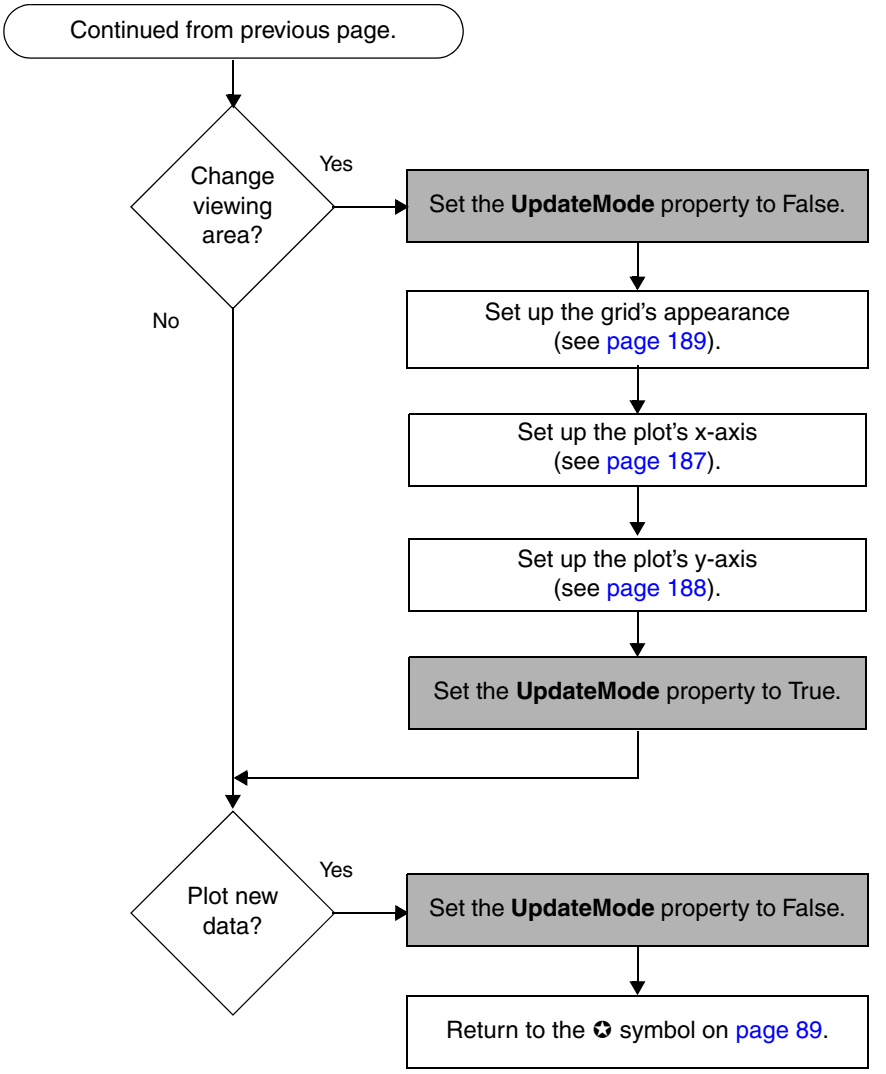

Continued from previous page.
Change viewing area?
Yes
No
Set the **UpdateMode** property to False.
Set up the grid's appearance (see page 189).
Set up the plot's x-axis (see page 187).
Set up the plot's y-axis (see page 188).
Set the **UpdateMode** property to True.
Plot new data?
Yes
Set the **UpdateMode** property to False.
Return to the ✪ symbol on page 89.

Figure 23: Plotting Data (cont.)

Software Architecture

Introduction. 92
System Operations 93
Analog and Digital I/O Operations 99
Counter/Timer Operations. 137
Simultaneous Operations 164
Plot Control Operations. 166

Introduction

This chapter provides conceptual information to describe the following operations provided by DTx-EZ:

- System operations, described starting on page 93;
- Analog and digital I/O operations, described starting on page 99;
- Counter/timer operations, described starting on page 137;
- Simultaneous operations, described starting on page 164; and
- Plot control operations, described starting on page 166.

Use this information with the reference information provided in DTx-EZ online help when programming your data acquisition boards; refer to page 13 for more information on launching this help file.

System Operations

DTx-EZ provides functions to perform the following general system operations:

- Initializing and specifying a board (this page),
- Specifying a subsystem (this page),
- Configuring a subsystem (page 95),
- Handling events (page 96),
- Handling errors (page 96), and
- Halting the operation (page 98).

The following subsections describe these operations in more detail.

Initializing and Specifying a Board

To perform any data acquisition operation, your application program must initialize the device driver for a specified board using the **Board** property. At run-time, you can use the **BoardList** property. This property lists the DT-Open Layers boards available in your system. Use the **numBoards** property to determine the number of DT-Open Layers boards currently installed in your system.

Once you have selected a board, you can specify a subsystem, as described in the next section.

Specifying a Subsystem

A *subsystem* refers to the major circuitry on a board. DTx-EZ defines the following subsystems:

- Analog input (A/D subsystem),
- Analog output (D/A subsystem),
- Digital input (DIN subsystem),
- Digital output (DOUT subsystem),
- Counter/timer (C/T subsystem), and
- Serial port (SRL subsystem).

Note: The SRL subsystem is provided for future use. It is not currently used by any DT-Open Layers compatible data acquisition device.

A board can have multiple *elements* of the same subsystem type. Each of these elements is a subsystem of its own and is identified by a subsystem type and element number. Element numbering is zero-based; that is, the first instance of the subsystem is called element 0, the second instance of the subsystem is called element 1, and so on. For example if two digital I/O ports are on your board, two DIN or DOUT subsystems are available, differentiated as element 0 and element 1.

Once you have selected a board, you must specify the subsystem/element using the **SubSystem** property. This property returns a *subsystem handle*, called hdass. To directly access a subsystem via the DT-Open Layers DLLs, you need one subsystem handle for each subsystem.

Alternately, you can use the **SubSysType** and **SubSysElement** properties at runtime to create a device-independent application.

With these two properties, you can set up the application to use a specific subsystem type and element on any selected device (for example, the first element of type A/D). In comparison, the **Subsystem** property simply selects the N^{th} subsystem on the board, which prevents device-independent operation.

If you are unsure of the subsystems on a device, use the **EnumSS** or the **GetDevCaps** method. **EnumSS** lists the names and types of elements for all subsystems supported by the specified device. **GetDevCaps** returns the number of elements for a specified subsystem type on a specified device. Additionally, during runtime, use the **numSubSystems** and **SubSystemList** properties to list the number of subsystems available on a DT-Open Layers board.

Once you have specified a subsystem/element, you can configure the subsystem and perform a data acquisition operation, as described in the following section.

Configuring a Subsystem

You configure a subsystem by setting its parameters or capabilities. For more information on the capabilities you can query and specify, refer to the following:

- For analog and digital I/O operations, refer to page 99;
- For the counter/timer operations, refer to page 137, and
- For simultaneous operations, refer to page 164.

Once you have set up the parameters appropriately for the operation you want to perform, use the **Config** method to configure the parameters before performing the operation.

Handling Events

The data acquisition board notifies your application of buffer movement and other activities by generating events.

Refer to DTx-EZ online help for more information on the events that can be generated.

Handling Errors

DTx-EZ Custom Controls produce two types of errors. The first is an OLE automation error that occurs when accessing one of the control's properties or methods. The second type of error occurs during the actual operation of the control's subsystem and is signaled by the OverrunError Event, UnderrunError Event, TriggerError event, or EventError Event.

Resolve the second error type by entering code in your control's event handler subroutine. Resolve an OLE automation error with one of the following routines (depending on your development environment):

From Visual Basic

```
   On Error GoTo DealWithIt
   DTAcq321.Start'start the subsystem
   Exit Sub
DealWithIt:
   if Err.Number = 440 then
   'OLE automation error occurred
      MsgBox "DTAcq321 Produced DT-Open Layers
      error:"+ CStr(DTAcq321.LastError) + ", " +
DTAcq321.LastErrDescription
   End If
```

From Visual C++

```
try
{
   m_DTAcq321.Start()//start the subsystem
}
catch(COleDispatchException* e)
{
   char myError[100];
   sprintf(myError,"DT-Open Layers Error: %ld ,
   %s", e->m_scError - DTACQ32_ERRORBASE,
   e->m_strDescription);
   AfxMessageBox(myError);
   e-Delete();
//delete what you have dealt with
```

DTx-EZ Visual Basic functions raise errors and you can deal with them in the following manner:

From Visual Basic

```
   On Error GoTo TrapIt
   ValidSamples = ValidSamples(hbuf)

'…
   Exit Sub
TrapIt:
   MsgBox Err.Source + " Error = " +
      CStr(Err.Number - vbObjectError) +
      ": " + Err.Description
End Sub
```

From Visual C++

```
ECODE ecode = olDmGetValidSamples(hbuf,
   &ulNumSamples);
if(ecode != OLNOERROR)
{
   char msg[100];
   olDmGetErrorString(ecode,msg,100);
   AfxMessageBox(msg);
```

Halting the Operation

When you are finished performing data acquisition operations, stop each subsystem with the **Stop** method. Then, release the simultaneous start list, if used, using the **ReleaseSimultaneousStartList** subroutine (for Visual Basic) or **olDaReleaseSSList** (for Visual C++).

Analog and Digital I/O Operations

DTx-EZ defines the following capabilities that you can query and/or specify for analog and/or digital I/O operations:

- Data encoding (this page),
- Resolution (page 100),
- Channels (including channel type, channel list, channel inhibit list, and synchronous digital I/O list) (page 100),
- Ranges (page 108),
- Gains (page 109),
- Filters (page 111),
- Data flow modes (page 112),
- Triggered scan mode (page 118),
- Clock sources (page 122),
- Trigger sources (page 124),
- Buffers (page 128), and
- DMA resources (page 135).

The following subsections describe these capabilities in more detail.

Data Encoding

For A/D and D/A subsystems only, DTx-EZ defines two data encoding types: binary and twos complement.

To determine the data encoding types supported by the subsystem, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_BINARY for binary data encoding or OLSSC_SUP_2SCOMP for twos complement data encoding. If this method returns a non-zero value, the capability is supported.

Use the **Encoding** property to specify the data encoding type.

Resolution

Different subsystems may support a number of software-programmable resolutions. To determine if the subsystem supports software-programmable resolution, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_SWRESOLUTION. If this method returns a non-zero value, the capability is supported.

To determine the number of resolution settings supported by the subsystem, use the **GetSSCaps** method, specifying the capability OLSSC_NUMRESOLUTION. To list the actual bits of resolution supported, use the **EnumSSCaps** function, specifying the OL_ENUM_RESOLUTION capability.

During runtime, use the **numResolutions** property to list the number of resolutions available to the subsystem; use the **ResolutionValues** property to list the actual resolutions available to the subsystem.

Use the **Resolution** property to specify the number of bits of resolution to use for the subsystem.

Channels

Each subsystem (or element of a subsystem type) can have multiple channels. To determine how many channels the subsystem supports, use the **GetSSCaps** method, specifying the OLSSC_NUMCHANNELS capability.

Specifying the Channel Type

DTx-EZ supports the following channel types:

- **Single-ended** –Use this configuration when you want to measure high-level signals, noise is insignificant, the source of the input is close to the device, and all the input signals are referred to the same common ground.

  To determine if the subsystem supports the single-ended channel type, use the **GetSSCaps** method, specifying the OLSSC_SUP_SINGLEENDED capability. If this method returns a non-zero value, the capability is supported.

  To determine how many single-ended channels are supported by the subsystem, use the **GetSSCaps** method, specifying the OLSSC_MAXSECHANS capability.

- **Differential** –Use this configuration when you want to measure low-level signals (less than 1 V), you are using an A/D converter with high resolution (> 12 bits), noise is a significant part of the signal, or common-mode voltage exists.
  To determine if the subsystem supports the differential channel type, use the **GetSSCaps** method, specifying the OLSSC_SUP_DIFFERENTIAL capability. If this method returns a non-zero value, the capability is supported.

  To determine how many differential channels are supported by the subsystem, use the **GetSSCaps** method, specifying the OLSSC_MAXDICHANS capability.

- Specify the channel type as differential for each channel using the **ChannelType** property.

Notes: For pseudo-differential analog inputs, specify the single-ended channel type; in this case, how you wire these signals determines the configuration. This option provides less noise rejection than the differential configuration, but twice as many analog input channels.

For older model boards, this setting is jumper-selectable and must be specified in the driver configuration dialog.

The channel list is not used to set the channel type.

The following subsections describe how to specify channels.

Specifying a Single Channel

The simplest way to acquire data from or output data to a single channel is to specify the channel for a single value operation; refer to page 112 for more information on single value operations.

You can also specify a single channel using a channel list, described in the next section.

Specifying One or More Channels

You acquire data from or output data to one or more channels using a channel list.

DTx-EZ provides features that allow you to group the channels in the list sequentially (either starting with 0 or with any other analog input channel) or randomly. In addition, DTx-EZ allows you to specify a single channel or the same channel more than once in the list. Your device, however, may limit the order in which you can enter channel in the channel list.

To determine how the channels can be ordered in the channel list for your subsystem, use the **GetSSCaps** method, specifying the OLSSC_RANDOM_CGL capability. If this method returns a non-zero value, the capability is supported; you can order the channels in the channel list in any order, starting with any channel. If this capability is not supported, use the **GetSSCaps** method, specifying the OLSSC_SUP_SEQUENTIAL_CGL capability. If this method returns a non-zero value, the capability is supported; you must order the channels in the channel list in sequential order, starting with any channel. If this capability is not supported, use the **GetSSCaps** method, specifying the OLSSC_SUP_ZEROSEQUENTIAL_CGL capability. If this method returns a non-zero value, the capability is supported; you must order the channels in the channel list in sequential order, starting with channel 0.

To determine if the subsystem supports simultaneous sample-and-hold mode use the **GetSSCaps** method, specifying the OLSSC_SUP_SIMULTANEOUS_SH capability. If this method returns a non-zero value, the capability is supported. You must enter at least two channels in the channel list. Generally, the first channel is the sample channel and the remaining channels are the hold channels.

The following subsections describe how to specify channels in a channel list.

Specifying the Channel List Size

To determine the maximum size of the channel list for the subsystem, use the **GetSSCaps** method, specifying the OLSSC_CGLDEPTH capability.

Use the **ListSize** property to specify the size of the channel list.

Note: The OLSSC_CGLDEPTH capability specifies the maximum size of the channel list, channel inhibit list, synchronous digital I/O list, and gain list.

Specifying the Channels in the Channel List

Use the **ChannelList** property to specify the channels in the channel list in the order you want to sample them or output data from them.

The channels are sampled or output in order from the first entry to the last entry in the channel list. Channel numbering is zero-based; that is, the first entry in the channel list is entry 0, the second entry is entry 1, and so on.

For example, if you want to sample channel 4 twice as frequently as channels 5 and 6, you could program the channel list as follows:

Channel-List Entry	Channel	Description
0	4	Sample channel 4.
1	5	Sample channel 5.
2	4	Sample channel 4 again.
3	6	Sample channel 6.

In this example, channel 4 is sampled first, followed by channel 5, channel 4 again, then channel 6.

Inhibiting Channels in the Channel List

If supported, you can set up a channel-inhibit list; this feature is useful if you want to discard values acquired from specific channels, as is typical in simultaneous sample-and-hold applications.

To determine if a subsystem supports a channel-inhibit list, use the **GetSSCaps** method, specifying the OLSSC_SUP_CHANNELLIST_INHIBIT capability. If this method returns a non-zero value, the capability is supported.

Using the **InhibitList** property, you can enable or disable inhibition for each entry in the channel list. If enabled, the acquired value is discarded after the channel entry is sampled; if disabled, the acquired value is stored after the channel entry is sampled.

Consider the following example:

Channel-List Entry	Channel	Channel Inhibit Value	Description
0	11	True	Sample channel 11 and discard the value.
1	10	False	Sample channel 10 and store the value.
2	9	True	Sample channel 9 and discard the value.
3	8	False	Sample channel 8 and store the value.

In this example, the values acquired from channels 11 and 9 are discarded and the values acquired from channels 10 and 8 are stored.

Specifying Synchronous Digital I/O Values in the Channel List

If supported, you can set up a synchronous digital I/O list; this feature is useful if you want to write a digital output value to dynamic digital output channels when an analog input channel is sampled.

To determine if the subsystem supports synchronous (dynamic) digital output operations, use the **GetSSCaps** method, specifying the OLSSC_SUP_SYNCHRONOUSDIGITALIO capability. If this method returns a non-zero value, the capability is supported.

Use the **SyncDIOUsage** property to enable or disable synchronous (dynamic) digital output operation for a specified subsystem.

Once you enable a synchronous digital output operation, specify the values to write to the synchronous (dynamic) digital output channels using the **DIOList** property for each entry in the channel list.

To determine the maximum digital output value that you can specify, use the **GetSSCaps** method, specifying the OLSSC_MAXDIGITALIOLIST_VALUE capability.

As each entry in the channel list is scanned, the corresponding value in the synchronous digital I/O list is output to the dynamic digital output channels. Consider the following example:

Channel-List Entry	Channel	Synchronous Digital I/O Value	Description
0	7	1	Sample channel 7 and output a value of 1 to the dynamic digital output channels.
1	5	1	Sample channel 5 and output a value of 1 to the dynamic digital output channels.
2	6	0	Sample channel 6 and output a value of 0 to the dynamic digital output channels.
3	4	0	Sample channel 4 and output a value of 0 to the dynamic digital output channels.

In this case, when channel 7 is sampled, a value of 1 is output to the dynamic digital output channels. When channel 5 is sampled, a value of 1 is output to the dynamic digital output channels. When channels 6 and 4 are sampled, a value of 0 is output to the dynamic digital output channels.

If your device had two dynamic digital output bits and a value of 1 is output (01 in binary format), a value of 1 is written to dynamic digital output bit 0 and a value of 0 is written to dynamic digital output bit 1. Similarly, if a value of 2 is output (10 in binary format), a value of 0 is written to dynamic digital output bit 0 and a value of 1 is written to dynamic digital output bit 1.

Note: If you are controlling sample-and-hold devices with these channels, you may need to program the first channel at the sample logic level and the following channels at the hold logic level; see your board/device driver documentation for details.

Ranges

The range capability applies to A/D and D/A subsystems only.

Depending on your subsystem, you can set the range for the entire subsystem or the range for each channel.

To determine if the subsystem supports the range-per-channel capability, use the **GetSSCaps** method, specifying the OLSSC_SUP_RANGEPERCHANNEL capability. If this method returns a non-zero value, the capability is supported.

To determine how many ranges the subsystem supports, use the **GetSSCaps** method, specifying the OLSSC_NUMRANGES capability.

To list the minimum and maximum ranges supported by the subsystem, use the **EnumSSCaps** function, specifying the OL_ENUM_RANGES capability.

During runtime, use the **MaxRange** and **MinRange** properties to return the maximum and minimum voltage values or a range setting. Use the **MaxRangeValues** and **MinRangeValues** properties to list the maximum and minimum voltage range values available to the subsystem. Use the **numRanges** property to list the number of available voltage ranges for the subsystem.

Use the **Range** property to specify the range for a subsystem.

Note: The channel list is not used to set the range for a channel.

For older board models, the range is jumper-selectable and must be specified in the driver configuration dialog.

Gains

The range divided by the gain determines the effective range for the entry in the channel list. For example, if your board provides a range of ±10 V and you want to measure a ±1.5 V signal, specify a range of ±10 V and a gain of 4; the effective input range for this channel is then ±2.5 V (10/4), which provides the best sampling accuracy for that channel.

The way you specify gain depends on how you specified the channels, as described in the following subsections.

Specifying the Gain for a Single Channel

The simplest way to specify gain for a single channel is to specify the gain in a single value operation; refer to page 112 for more information on single value operations.

You can also specify the gain for a single channel using a gain list, described in the next section.

Specifying the Gain for One or More Channels

You can specify the gain for one or more channels using a gain list. The gain list parallels the channel list. (The two lists together are often referred to as the channel-gain list or CGL.)

To determine if the subsystem supports programmable gain, use the **GetSSCaps** method, specifying the OLSSC_SUP_PROGRAMGAIN capability. If this method returns a non-zero value, the capability is supported.

To determine how many gains the subsystem supports, use the **GetSSCaps** method, specifying the OLSSC_NUMGAINS capability.

To list the gains supported by the subsystem, use the **EnumSSCaps** function, specifying the OL_ENUM_GAINS capability.

During runtime, to list the gains available to the subsystem use the **GainValues** property; to determine the number of gains available to the subsystem, use the **numGains** property.

You specify the gain for each entry in the channel list using the **GainList** property.

Consider the following example:

Channel-List Entry	Channel	Gain	Description
0	5	2	Sample channel 5 using a gain of 2.
1	6	4	Sample channel 6 using a gain of 4.
2	7	1	Sample channel 7 using a gain of 1.

In this example, a gain of 2 is applied to channel 5, a gain of 4 is applied to channel 6, and a gain of 1 is applied to channel 7.

Note: If your subsystem does not support programmable gain, enter a value of 1 for all entries.

If your subsystem does not support the gain-per-channel capability, set all entries in the gain list to the same value.

Filters

This capability applies to A/D subsystems only.

Depending on your subsystem, you can specify a filter for each channel. To determine if the subsystem supports a filter for each channel, use the **GetSSCaps** method, specifying the OLSSC_SUP_FILTERPERCHAN capability. If this method returns a non-zero value, the capability is supported.

To determine how many filters the subsystem supports, use the **GetSSCaps** method, specifying the OLSSC_NUMFILTERS capability.

To list the cut-off frequency of all filters supported by the subsystem, use the **EnumSSCaps** method, specifying the OL_ENUM_FILTERS capability.

During runtime, to list the filters available to the subsystem use the **FilterValues** property; to determine the number of filters available to the subsystem, use the **numFilters** property.

If the subsystem supports filtering per channel, specify the filter for each channel using the **FilterList** property. The filter is equal to or greater than a cut-off frequency that you supply.

Note: The channel list is not used to set the filter for a channel.

If the subsystem supports more than one filter but does not support a filter per channel, the filter specified for channel 0 is used for all channels.

Data Flow Modes

DTx-EZ defines the following data flow modes for A/D, D/A, C/T, DIN, and DOUT subsystems:

- Single value (this page), and
- Continuous (post-trigger, pre-trigger, and about-trigger) (page 113).

The following subsections describe these data flow modes in detail.

Single-Value Operations

Single value operations are the simplest to use but offer the least flexibility and efficiency. In a single value operation, a single data value is read or written at a time. The data is returned immediately.

To determine if the subsystem supports single value operations, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_SINGLEVALUE. If this method returns a non-zero value, the capability is supported.

Specify the operation mode as single value (1) using the **DataFlow** property.

For a single value operation, you can specify the data encoding, resolution, channel type, range, and filter, if supported, for the specified channel using the specified gain. You cannot specify other parameters, such as a channel-gain list, clock source, trigger source, DMA channel, or buffer.

Single value operations stop automatically when finished; you cannot stop a single value operation manually.

Once you have set up the parameters for a single value operation, use the **GetSingleValue** method to acquire a single analog or digital

value; use the **PutSingleValue** method to output a single analog or digital value.

Continuous Operations

For a continuous operation, you can specify any supported subsystem capability, including a channel-gain list, clock source, trigger source, pre-trigger source, retrigger source, DMA channel, and buffer.

Call the **Start** method to start a continuous operation.

To stop a continuous operation, perform either an orderly stop using the **Stop** method or an abrupt stop using the **Abort** or **Reset** method.

In an orderly stop (**Stop** method), the board finishes acquiring the specified number of samples, stops all subsequent acquisition, and transfers the acquired data to a buffer on the done queue; all subsequent triggers or retriggers are ignored.

In an abrupt stop (**Abort** method), the board stops acquiring samples immediately; the acquired data is transferred to a buffer and put on the done queue; however, the buffer may not be completely filled. All subsequent triggers or retriggers are ignored.

The **Reset** method reinitializes the subsystem after stopping it abruptly. (Refer to page 128 for more information on buffers and queues.)

Note: For analog output operations, you can also stop the operation by not sending new data to the board. The operation stops when no more data is available.

Some subsystems also allow you to pause the operation using the **Pause** method and to resume the paused operation using the **Continue** method. To determine if pausing is supported, use the **GetSSCaps** method, specifying the OLSSC_SUP_PAUSE capability. If this method returns a non-zero value, the capability is supported.

The following continuous modes are supported by DTx-EZ: continuous (post-trigger), continuous pre-trigger, and continuous about-trigger. These modes are described in the following subsections.

Continuous (Post-Trigger) Mode

Use continuous (post-trigger) when you want to acquire or output data continuously when a trigger occurs.

To determine if the subsystem supports continuous (post-trigger) operations, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_CONTINUOUS. If this method returns a non-zero value, the capability is supported.

For continuous (post-trigger) mode, specify the operation mode as continuous (0) using the **DataFlow** property.

Use the **Trigger** property to specify the trigger source that starts the operation. Refer to page 124 for more information on supported trigger sources.

When the post-trigger occurrence is detected, the board cycles through the channel list, acquiring and/or outputting the value for each entry in the channel list; this process is defined as a scan. The board then wraps to the start of the channel list and repeats the process continuously until either the allocated buffers are filled or you stop the operation. Refer to page 102 for more information on channel lists; refer to page 128 for more information on buffers.

Figure 24 illustrates continuous post-trigger mode using a channel list of three entries: channel 0, channel 1, and channel 2. In this example, post-trigger analog input data is acquired on each clock pulse of the A/D sample clock; refer to page 122 for more information on clock sources. The board wraps to the beginning of the channel list and repeats continuously.

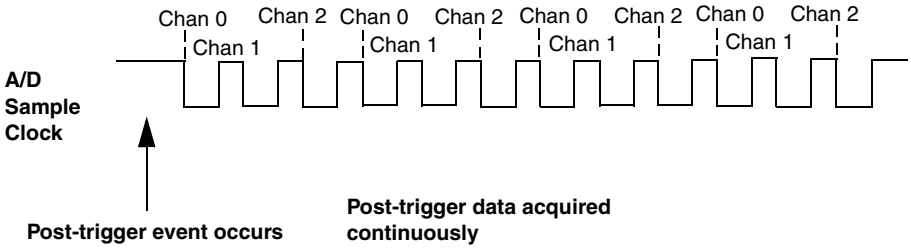


Figure 24: Continuous Post-Trigger Mode

5

Continuous Pre-Trigger Mode

Use continuous pre-trigger mode when you want to acquire data before a specific external event occurs.

To determine if the subsystem supports continuous pre-trigger mode, use the **GetSSCaps** method, specifying the OLSSC_SUP_CONTINUOUS_PRETRIG capability. If this method returns a non-zero value, the capability is supported.

Specify the operation mode as continuous pre-trigger (4) using the **DataFlow** property.

Pre-trigger acquisition starts when the device detects the pre-trigger source and stops when the board detects an external post-trigger source, indicating that the first post-trigger sample was acquired (this sample is ignored).

Use the **PreTrigger** property to specify the trigger source that starts the pre-trigger operation (generally this is a software trigger). Specify the post-trigger source that stops the operation using the **Trigger** property. Refer to page 124 and to your board/driver documentation for supported sources.

Figure 25 illustrates continuous pre-trigger mode using a channel list of three entries: channel 0, channel 1, and channel 2. In this example, pre-trigger analog input data is acquired on each clock pulse of the A/D sample clock; refer to page 122 for more information on clock sources. The board wraps to the beginning of the channel list and the acquisition repeats continuously until the post-trigger event occurs. As your buffers are filled and placed on the done queue, PreTriggerBufferDone events occur. When the post-trigger action occurs, acquisition stops, and a QueueStopped event occurs.

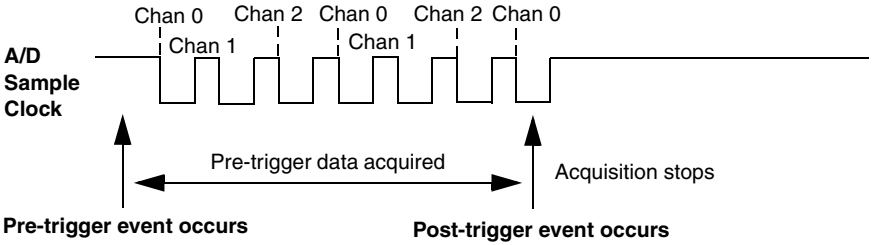


Figure 25: Continuous Pre-Trigger Mode

Continuous About-Trigger Mode

Use continuous about-trigger mode when you want to acquire data both before and after a specific external event occurs. This operation is equivalent to doing both a pre-trigger and a post-trigger acquisition.

To determine if the subsystem supports continuous about-trigger mode, use the **GetSSCaps** method, specifying the OLSSC_SUP_CONTINUOUS_ABOUTTRIG capability. If this method returns a non-zero value, the capability is supported.

Specify the operation mode as continuous about-trigger (5) using the **DataFlow** property.

The about-trigger acquisition starts when the board detects the pre-trigger source. When it detects an external post-trigger source, the board stops acquiring pre-trigger data and starts acquiring post-trigger data.

Use the **PreTrigger** property to specify the pre-trigger source that starts the pre-trigger operation (this is generally a software trigger) and the **Trigger** property to specify the trigger source that stops the pre-trigger acquisition and starts the post-trigger acquisition. Refer to page 124 and to your board/driver documentation for supported pre-trigger and post-trigger sources.

The about-trigger operation stops when the specified number of post-trigger samples has been acquired or when you stop the operation.

Figure 26 illustrates continuous about-trigger mode using a channel list of three entries: channel 0, channel 1, and channel 2. In this example, pre-trigger analog input data is acquired on each clock pulse of the A/D sample clock. The board wraps to the beginning of the channel list and the acquisition repeats continuously until the post-trigger event occurs. When the post-trigger event occurs, post-trigger acquisition begins on each clock pulse of the A/D sample clock; refer to page 122 for more information on clock sources. The board wraps to the beginning of the channel list and acquires post-trigger data continuously.

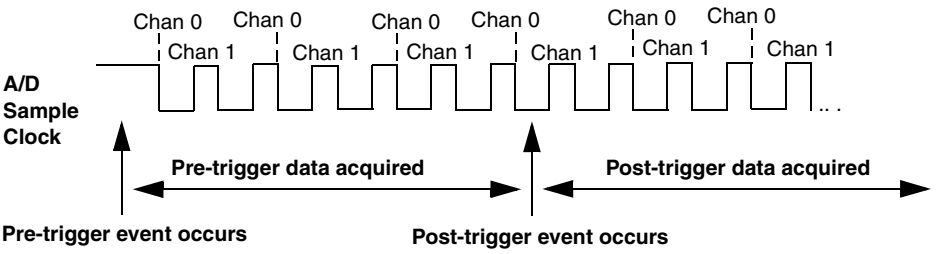


Figure 26: Continuous About-Trigger Mode

As your buffers fill during the pre-trigger state, the buffers are placed on the done queue, and a PreTriggerBufferDone event occurs. Buffers filled during the post-trigger state are placed on the done queue, and a BufferDone event occurs. Check the valid number of samples in the buffers before using them, since the last buffer filled prior to switching to the post-trigger state may be only partially full.

Triggered Scan Mode

In triggered scan mode, the board scans the entries in a channel-gain list a specified number of times when it detects the specified trigger source, acquiring the data for each entry that is scanned.

To determine if the subsystem supports triggered scan mode, use the **GetSSCaps** method, specifying the OLSSC_SUP_TRIGSCAN capability. If this method returns a non-zero value, the capability is supported. Note that you cannot use triggered scan mode with single value operations.

To enable (or disable) triggered scan mode, use the **TriggeredScan** property.

To determine the maximum number of times that the board can scan the channel-gain list per trigger, use the **GetSSCaps** method, specifying the OLSSC_MAXMULTISCAN capability.

Use the **MultiscanCount** property to specify the number of times to scan the channel-gain list per trigger.

DTx-EZ defines the following retrigger modes for a triggered scan; these retrigger modes are described in the following subsections:

- Scan-per-trigger (page 119),
- Internal retrigger (page 120), and
- Retrigger extra (page 121).

Note: If your device driver supports it, retrigger extra is the preferred triggered scan mode.

Scan-Per-Trigger Mode

Use scan-per-trigger mode if you want to accurately control the period between conversions of individual channels and retrigger the scan based on an internal or external event. In this mode, the retrigger source is the same as the initial trigger source.

To determine if the subsystem supports scan-per-trigger mode, use the **GetSSCaps** method, specifying the OLSSC_SUP_RETRIGGER_SCAN_PER_TRIGGER capability. If this method returns a non-zero value, the capability is supported.

Specify the retrigger mode as OL_TRIGGER_SCAN_PER_TRIGGER using the **RetriggerMode** property.

When it detects an initial trigger (post-trigger source only), the board scans the channel-gain list a specified number of times (determined by the **MultiscanCount** property), then stops. When the external retrigger occurs, the process repeats.

The conversion rate of each channel in the scan is determined by the frequency of the A/D sample clock; refer to page 122 for more information on clock sources. The conversion rate of each scan is determined by the period between retriggers; therefore, it cannot be accurately controlled. The board ignores external triggers that occur while it is acquiring a scan of data. Only retrigger events that occur when the board is waiting for a trigger are detected and acted on. Some boards may generate a TriggerError event.

Internal Retrigger Mode

Use internal retrigger mode if you want to accurately control both the period between conversions of individual channels in a scan and the period between each scan.

To determine if the subsystem supports internal retrigger mode, use the **GetSSCaps** method, specifying the OLSSC_SUP_RETRIGGER_INTERNAL capability. If this method returns a non-zero value, the capability is supported.

Specify the retrigger mode as OL_RETRIGGER_INTERNAL using the **RetriggerMode** property.

The conversion rate of each channel in the scan is determined by the frequency of the A/D sample clock; refer to page 122 for more information on clock sources. The conversion rate between scans is determined by the frequency of the internal retrigger clock on the board. You specify the frequency on the internal retrigger clock using the **RetriggerFreq** property.

When it detects an initial trigger (pre-trigger source or post-trigger source), the board scans the channel-gain list a specified number of times (determined by the **MultiscanCount** property), then stops. When the internal retrigger occurs, determined by the frequency of the internal retrigger clock, the process repeats.

It is recommended that you set the retrigger frequency as follows:

$$\text{Min. Retrigger Period} = \frac{\text{\# of CGL entries} \times \text{\# of CGLs per trigger}}{\text{A/D sample clock frequency}} + 2\ \mu s$$

$$\text{Max. Retrigger Frequency} = \frac{1}{\text{Min. Retrigger Period}}$$

For example, if you are using 512 channels in the channel-gain list (CGL), scanning the channel-gain list 256 times every trigger or retrigger, and using an A/D sample clock with a frequency of 1 MHz, set the maximum retrigger frequency to 7.62 Hz, since

$$7.62\ \text{Hz} = \frac{1}{\frac{(512 * 256)}{1\ \text{MHz}} + 2\ \mu s}$$

Retrigger Extra Mode

Use retrigger extra mode if you want to accurately control the period between conversions of individual channels and retrigger the scan on a specified retrigger source; the retrigger source can be any of the supported trigger sources.

To determine if the subsystem supports retrigger extra mode, use the **GetSSCaps** method, specifying the OLSSC_SUP_RETRIGGER_EXTRA capability. If this method returns a non-zero value, the capability is supported.

Specify the retrigger mode as OL_RETRIGGER_EXTRA using the **RetriggerMode** property.

Use the **ReTrigger** property to specify the retrigger source. Refer to page 124 and to your board/device driver documentation for supported triggering sources.

The conversion rate of each channel in the scan is determined by the frequency of the A/D sample clock; refer to page 122 for more information on clock sources. The conversion rate of each scan is determined by the period between retriggers. For external retriggers, the period between retriggers cannot be accurately controlled. For internal retriggers, specify the period between retriggers using the **RetriggerFreq** property (see page 120). The board ignores triggers that occur while it is acquiring data. Only retrigger events that occur when the board is waiting for a trigger are detected and acted on. Some boards may generate a TriggerError event.

Clock Sources

DTx-EZ defines internal (this page), external (page 123), and extra (page 124) clock sources, described in the following subsections. Note that you cannot specify a clock source for single value operations.

Internal Clock Source

The internal clock is the clock source on the board that paces data acquisition or output for each entry in the channel-gain list.

To determine if the subsystem supports an internal clock, use the **GetSSCaps** method, specifying the OLSSC_SUP_INTCLOCK capability. If this method returns a non-zero value, the capability is supported.

Specify the clock source as internal (0) using the **ClockSource** property. Then, use the **Frequency** property to specify the frequency at which to pace the operation.

To determine the maximum frequency that the subsystem supports, use the **GetSSCapsEx** method, specifying the OLSSCE_MAXTHROUGHPUT capability. To determine the minimum frequency that the subsystem supports, use the **GetSSCapsEx** method, specifying the OLSSCE_MINTHROUGHPUT capability.

Note: According to sampling theory (Nyquist Theorem), you should specify a frequency for an A/D signal that is at least twice as fast as the input's highest frequency component. For example, to accurately sample a 20 kHz signal, specify a sampling frequency of at least 40 kHz. Doing so avoids an error condition called *aliasing*, in which high frequency input components erroneously appear as lower frequencies after sampling.

External Clock Source

The external clock is a clock source attached to the board that paces data acquisition or output for each entry in the channel-gain list. This clock source is useful when you want to pace at rates not available with the internal clock or if you want to pace at uneven intervals.

To determine if the subsystem supports an external clock, use the **GetSSCaps** method, specifying the OLSSC_SUP_EXTCLOCK capability. If this method returns a non-zero value, the capability is supported.

Specify the clock source as external using the **ClockSource** property. Then, use the **ClockDivider** property to specify the clock divider used to determine the frequency at which to pace the operation; the clock input source divided by the clock divider determines the frequency of the clock signal.

To determine the maximum clock divider that the subsystem supports, use the **GetSSCapsEx** method, specifying the OLSSCE_MAXCLOCKDIVIDER capability. To determine the minimum clock divider that the subsystem supports, use the **GetSSCapsEx** method, specifying the OLSSCE_MINCLOCKDIVIDER capability.

Extra Clock Source

Your device driver may define extra clock sources that you can use to pace acquisition or output operations.

To determine how many extra clock sources are supported by your subsystem, use the **GetSSCaps** method, specifying the OLSSC_NUMEXTRACLOCKS capability. Refer to your board/driver documentation for a description of the extra clock sources.

The extra clock sources may be internal or external. Refer to the previous sections for information on how to specify internal and external clocks and their frequencies or clock dividers.

Trigger Sources

DTx-EZ defines the following trigger sources:

- Software (internal) trigger (this page),
- External digital (TTL) trigger (page 125),
- External analog threshold (positive) trigger (page 126),
- External analog threshold (negative) trigger (page 126),
- Analog event trigger (page 127),
- Digital event trigger (page 127),
- Timer event trigger (page 127), and
- Extra trigger (page 127).

To specify a post-trigger source, use the **Trigger** property; refer to page 114 for more information. To specify a pre-trigger source, use the **PreTrigger** property; see page 115 for more information. To specify a retrigger source, use the **ReTrigger** property; see page 121 for more information.

The following subsections describe these trigger sources. Note that you cannot specify a trigger source for single value operations.

Software (Internal) Trigger Source

A software trigger occurs when you start the operation; internally, the computer writes to the board to begin the operation.

To determine if the subsystem supports a software trigger, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_SOFTTRIG. If this method returns a non-zero value, the capability is supported.

External Digital (TTL) Trigger Source

An external digital trigger is a digital (TTL) signal attached to the device.

To determine if the subsystem supports an external digital trigger, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_EXTERNTRIG. If this method returns a non-zero value, the capability is supported.

External Analog Threshold (Positive) Trigger Source

An external analog threshold (positive) trigger is generally either an analog signal from an analog input channel or an external analog signal attached to the device. An analog trigger occurs when the device detects a transition from a negative to positive value that crosses a threshold value. The threshold level is generally set using a D/A subsystem on the device.

To determine if the subsystem supports analog threshold triggering (positive polarity), use the **GetSSCaps** method, specifying the capability OLSSC_SUP_THRESHTRIGPOS. If this method returns a non-zero value, the capability is supported.

Refer to your board/device driver documentation for a description of this trigger source.

External Analog Threshold (Negative) Trigger Source

An external analog threshold (negative) trigger is generally either an analog signal from an analog input channel or an external analog signal attached to the device. An analog trigger event occurs when the device detects a transition from a positive to negative value that crosses a threshold value. The threshold level is generally set using a D/A subsystem on the device.

To determine if the subsystem supports analog threshold triggering (negative polarity), use the **GetSSCaps** method, specifying the capability OLSSC_SUP_THRESHTRIGNEG. If this method returns a non-zero value, the capability is supported.

Refer to your board/device driver documentation for a description of this trigger source.

Analog Event Trigger Source

For this trigger source, a trigger is generated when an analog event occurs. To determine if the subsystem supports an analog event trigger, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_ANALOGEVENTTRIG. If this method returns a non-zero value, the capability is supported.

Digital Event Trigger Source

For this trigger source, a trigger is generated when a digital event occurs. To determine if the subsystem supports a digital event trigger, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_DIGITALEVENTTRIG. If this method returns a non-zero value, the capability is supported.

Timer Event Trigger Source

For this trigger source, a trigger is generated when a counter/timer event occurs. To determine if the subsystem supports a timer event trigger, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_TIMEREVENTTRIG. If this method returns a non-zero value, the capability is supported.

Extra Trigger Source

Extra trigger sources may be defined by your device driver. To determine how many extra triggers are supported by the subsystem, use the **GetSSCaps** method, specifying the capability OLSSC_NUMEXTRATRIGGERS. Refer to your board/driver documentation for a description of these triggers.

The extra trigger sources may be internal or external. Refer to the previous sections for information on how to specify internal and external triggers.

Buffers

The buffering capability usually applies to A/D and D/A subsystems only. Note that you cannot use a buffer with single value operations.

A data buffer is a memory location that you allocate in host memory. This memory location is used to store data for continuous input and output operations.

To determine if the subsystem supports buffers, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_BUFFERING. If this method returns a non-zero value, the capability is supported.

Buffers are stored on one of three queues: the ready queue (this page), the inprocess queue (page 130), or the done queue (page 131). These queues are described in more detail in the following subsections.

Note: In these subsections, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ name appears in italics following the Visual Basic name.

Ready Queue

For input operations, the ready queue holds buffers that are empty and ready for input. For output operations, the ready queue holds buffers that you have filled with data and that are ready for output.

Allocate the buffers using the **AllocBuffer** (***olDmAllocBuffer***) or the **CallocBuffer** (***olDmCallocBuffer***) function. **AllocBuffer** (***olDmAllocBuffer***) allocates a buffer of samples, where each sample is 2 bytes; **CallocBuffer** (***olDmCallocBuffer***) allocates a buffer of samples of a specified size.

For analog input operations, it is recommended that you allocate a minimum of three buffers; for analog output operations, you can allocate one or more buffers. The size of the buffers should be at least as large as the sampling or output rate; for example, if you are using a sampling rate of 100 ksamples/s (100 kHz), specify a buffer size of 100,000 samples.

Once you have allocated the buffers (and, for output operations, filled them with data and set the valid number of samples), put the buffers on the ready queue using the **Queue** property.

For example, assume that you are performing an analog input operation, that you allocated three buffers, and that you put these buffers on the ready queue. The queues appear on the ready queue as shown in Figure 27.

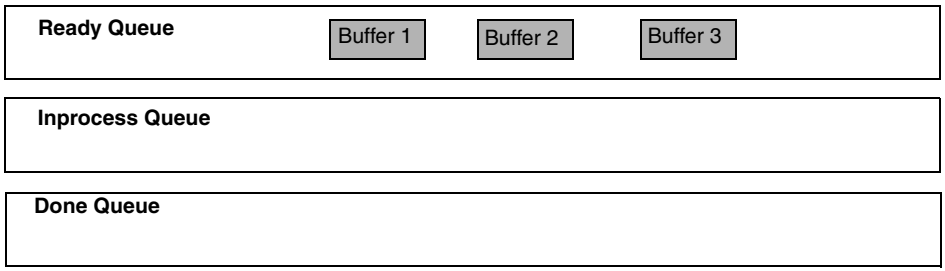


Figure 27: Example of the Ready Queue

Inprocess Queue

When you start a continuous (post-trigger, pre-trigger, or about-trigger) operation, the data acquisition board takes the first available buffer from the ready queue and places it on the inprocess queue.

The inprocess queue holds the buffer that the specified data acquisition board is currently filling (for input operations) or outputting (for output operations). The buffer is filled or emptied at the specified clock rate.

Continuing with the previous example, when you start the analog input operation, the driver takes the first available buffer (Buffer 1, in this case), puts it on the inprocess queue, and starts filling it with data. The queues appear as shown in Figure 28.

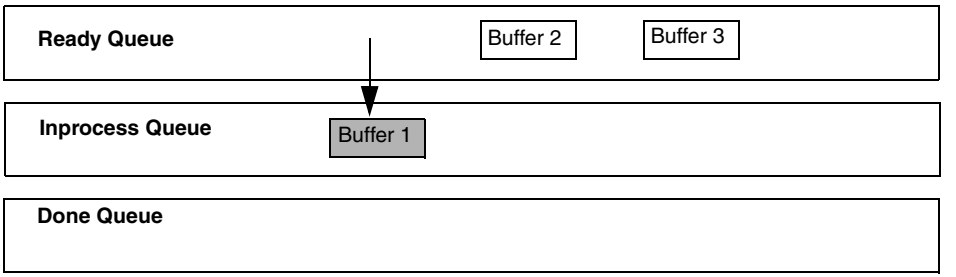


Figure 28: Example of the Inprocess Queue

If you want to transfer data from a partially-filled buffer, you can use the **FlushFromBufferInProcess** (***olDaFlushFromBufferInprocess***) subroutine to transfer data from the buffer on an inprocess queue to another buffer you allocate, if this capability is supported. Typically, you would use this function when your data acquisition operation is running slowly.

To determine if the subsystem supports transferring data from a buffer on the inprocess queue, use the **GetSSCaps** method, specifying the OLSSC_SUP_INPROCESS_FLUSH capability. If this method returns a non-zero value, this capability is supported.

Done Queue

Once the data acquisition board has filled the buffer (for input operations) or emptied the buffer (for output operations), the buffer is moved from the inprocess queue to the done queue. Then, either the BufferDone event is generated when the buffer contains post-trigger data, or in the case of pre- and about-trigger acquisitions, a PreTrigBufferDone event is generated when the buffer contains pre-trigger data.

Note: When the pre-trigger acquisition operation completes or you stop an acquisition, the QueueStopped event is also generated.

Continuing with the previous example, the queues appear as shown in Figure 29 when you get the first BufferDone event.

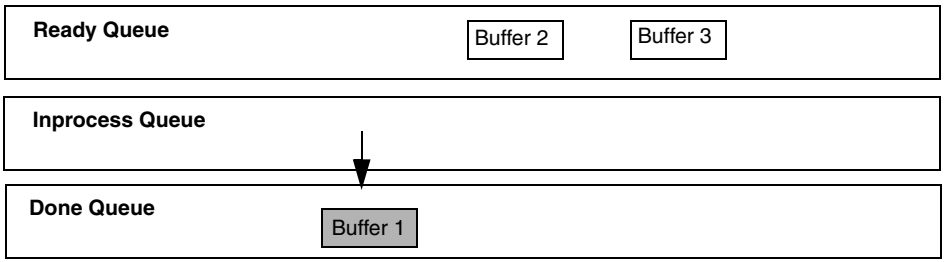


Figure 29: Example of the Done Queue

Then, the driver moves Buffer 2 from the ready queue to the inprocess queue and starts filling it with data. When Buffer 2 is filled, Buffer 2 is moved to the done queue and another BufferDone event is generated.

The driver then moves Buffer 3 from the ready queue to the inprocess queue and starts filling it with data. When Buffer 3 is filled, Buffer 3 is moved to the done queue and another BufferDone event is generated. Figure 30 shows how the buffers are moved.

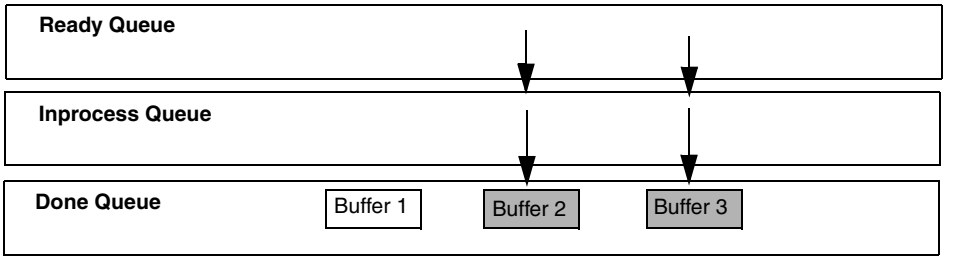


Figure 30: How Buffers are Moved to the Done Queue

If you transferred data from an inprocess queue to a new buffer using the **FlushFromBufferInprocess** (***olDaFlushFromBufferInprocess***) subroutine, the new buffer is put on the done queue for your application to process. When the buffer on the inprocess queue finishes being filled, this buffer is also put on the done queue; the buffer contains only the samples that were not previously transferred.

Buffer and Queue Management

Each time it gets a BufferDone event, your application program should remove the buffers from the done queue using the **Queue** property.

Your application program can then process the data in the buffer. For an input operation, you can copy the data from the buffer to an array in your application program using the **CopyFromBuffer** (***olDmGetBufferPtr***) subroutine. For continuously-paced analog output operations, you can fill the buffer with new output data using the **CopyToBuffer** (***olDmGetBufferPtr***) subroutine.

When you are finished processing the data, you can put the buffer back on the ready queue using the **Queue** property if you want your operation to continue.

For example, assume that you processed the data from Buffer 1 and put it back on the ready queue. The queues would appear as shown in Figure 31.

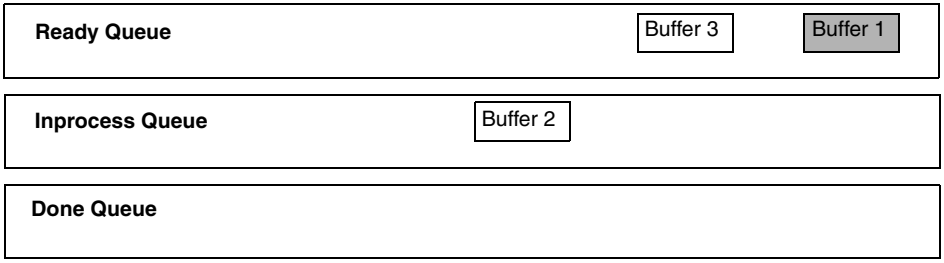


Figure 31: Putting Buffers Back on the Ready Queue

When the data acquisition operation is finished, use the **Flush** method to transfer any data buffers left on the subsystem's ready queue to the done queue.

Once you have processed the data in the buffers, free the buffers from the memory using the **FreeBuffer** (***olDmFreeBuffer***) subroutine.

Buffer Wrap Modes

Most Keithley data acquisition boards can provide gap-free data, meaning no samples are missed when data is acquired or output. You can acquire gap-free data by manipulating data buffers so that no gaps exist between the last sample of the current buffer and the first sample of the next buffer.

Note: The number of DMA channels, number of buffers, and buffer size are critical to the board's ability to provide gap-free data. It is also critical that the application process the data in a timely fashion.

If you want to acquire gap-free input data, it is recommended that you specify a buffer wrap mode of *none* (0) using the **WrapMode** property. When a buffer wrap mode of none is selected, if you process the buffers and put them back on the ready queue in a timely manner, the operation continues indefinitely. When no buffers are available on the ready queue, the operation stops, and a QueueDone event is generated.

If you want to continuously reuse the buffers in the queues and you are not concerned with gap-free data, specify *multiple* buffer wrap mode (1) using the **WrapMode** property. When multiple wrap mode is selected and no buffers are available on the ready queue, the driver moves the oldest buffer from the done queue to the inprocess queue (regardless of whether you have processed its data), and overwrites the data in the buffer. This process continues indefinitely unless you stop it. When it reuses a buffer on the done queue, the driver generates a BufferReused event.

If you want to perform gap-free waveform generation analog output operations, specify *single buffer* wrap mode (2) using the **WrapMode** property**.** When single wrap mode is specified, a single buffer is reused continuously. In this case, the driver moves the buffer from the ready queue to the inprocess queue and outputs the data from the buffer. However, when the buffer is emptied, the driver (or board) reuses the data and continuously outputs it. This process repeats indefinitely until you stop it. When you stop the operation, the buffer is moved to the done queue. No messages are posted in this mode until you stop the operation.

To determine the buffer wrap modes available for the subsystem, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_WRPSINGLE (for single wrap mode) or OLSSC_SUP_WRPMULTIPLE (for multiple wrap mode). If this method returns a non-zero value, the capability is supported.

DMA Resources

You cannot use DMA or interrupt resources for single value operations.

To determine if your subsystem supports interrupt resources, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_INTERRUPT. If this method returns a non-zero value, the capability is supported.

Note: If supported, all DT-Open Layers boards use interrupt resources.

Generally, you specify interrupt resources on the board itself and in the driver configuration dialog.

To determine if the subsystem supports DMA resources, use the **GetSSCaps** method, specifying the capability OLSSC_NUMDMACHANS to determine how many DMA channels are supported. If supported, these channels must be specified in the driver configuration dialog. In addition, specify OLSSC_SUP_GAPFREE_NODMA (for gap free data using no DMA channels), OLSSC_SUP_GAPFREE_SINGLEDMA (for gap free data using one DMA channel), or OLSSC_SUP_GAPFREE_DUALDMA (for gap free data using two DMA channels). If this method returns a non-zero value, the capability is supported.

Use the **DmaUsage** property to specify the number of DMA channels to use.

Note: DMA channels are a limited resource and the request may not be honored if the requested number of channels is unavailable. For example, suppose that a device that supports both A/D and D/A subsystems provides hardware for two DMA channels, and that one DMA channel is currently allocated to the A/D subsystem. In this case, a request to the D/A subsystem to use two DMA channels will fail.

Counter/Timer Operations

Each user counter/timer channel accepts a clock input signal and gate input signal and outputs a clock output signal (also called a pulse output signal), as shown in Figure 32.

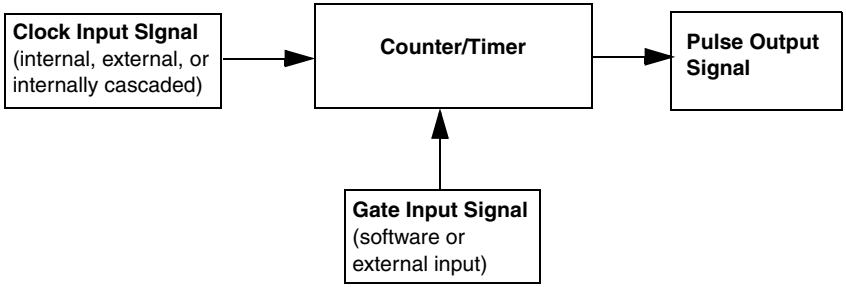


Figure 32: Counter/Timer Channel

Each counter/timer channel corresponds to a counter/timer (C/T) subsystem. To specify the counter to use in software, specify the appropriate C/T subsystem. For example, counter 0 corresponds to C/T subsystem element 0; counter 3 corresponds to C/T subsystem element 3.

DTx-EZ defines the following capabilities that you can query and/or configure for counter/timer operations:

- Counter/timer operation mode (page 138);
- Clock source (page 154);
- Gate type (page 157); and
- Pulse output type, output duty cycle, and width (page 162).

The following subsections describe these capabilities in more detail.

Counter/Timer Operation Mode

DTx-EZ supports the following counter/timer operations:

- Event counting (this page),
- Frequency measurement (page 140),
- Rate generation (continuous pulse output) (page 144),
- One-shot (page 148), and
- Repetitive one-shot (page 151).

The following subsections describe these counter/timer operations.

Event Counting

Use event counting mode to count events from the counter's associated clock input source.

To determine if the subsystem supports event counting, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_CTMODE_COUNT. If this method returns a non-zero value, the capability is supported.

To specify an event counting operation, use the **CTMode** property, specifying the count events (0) parameter.

Specify the C/T clock source for the operation. In event counting mode, an external C/T clock source is more useful than the internal C/T clock source; refer to page 154 for more information on specifying the C/T clock source.

Also specify the gate type that enables the operation; refer to page 157 for more information on specifying the gate type.

Start an event counting operation using the **Start** method. To read the current number events counted, use the **CTReadEvents** method.

To stop the event counting operation, call the **Stop**, **Abort**, or **Reset** method; **Reset** stops the operation and reinitializes the subsystem after stopping it.

Some subsystems also allow you to pause the operation using the **Pause** method and then resume the paused operation using the **Continue** method. To determine if pausing is supported, use the **GetSSCaps** method, specifying the OLSSC_SUP_PAUSE capability. If this method returns a non-zero value, the capability is supported.

Figure 33 shows an example of an event counting operation. In this example the gate type is low level.

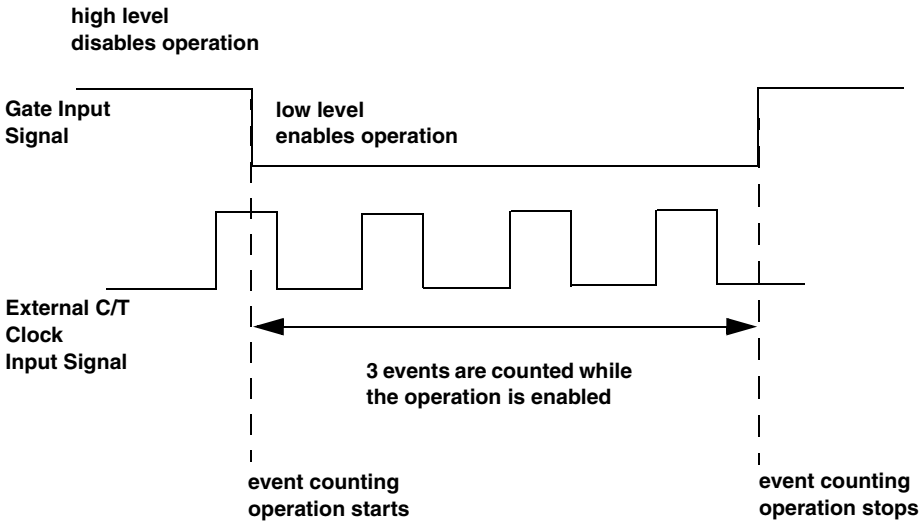


Figure 33: Example of Event Counting

Frequency Measurement

You can also use event counting mode to measure the frequency of the clock input signal for the counter, since frequency is the number events divided by a specified duration.

To determine if the subsystem supports event counting (and therefore, frequency measurement), use the **GetSSCaps** method, specifying the capability OLSSC_SUP_CTMODE_COUNT. If this method returns a non-zero value, the capability is supported.

You can perform a frequency measurement operation in one of two ways: using the Windows timer to specify the duration (this page) or using a pulse of a known duration as the gate input signal to a counter/timer configured for event counting mode (page 142). The following subsections describe these ways of measuring frequency.

Using the Windows Timer

To perform a frequency measurement operation on a single C/T subsystem using the Windows timer to specify the duration, perform the following steps:

1. Use the **CTMode** property, specifying the count events (0) parameter.
2. Specify the input clock source using the **ClockSource** property. In frequency measurement mode, an external C/T clock source is more useful than the internal C/T clock source; refer to page 154 for more information on the external C/T clock source.
3. Use the **GateType** property, specifying the no gate type parameter (0), to set the gate type to software.
4. Use the **MeasureFrequency** method to specify the duration of the Windows timer (which has a resolution of 1 ms) and to start the frequency measurement operation.

Frequency is determined using the following equation:

$$\text{Frequency} = \frac{\text{Number of Events}}{\text{Duration of the Windows Timer}}$$

When the operation is complete, the MeasureDone event is generated. Use the LongtoFreq (IParam) macro, described in DTx-EZ online help, to get the measured frequency value.

Figure 34 shows an example of a frequency measurement operation. Three events are counted from the clock input signals during a duration of 300 ms. The frequency is 10 Hz (3/.3).

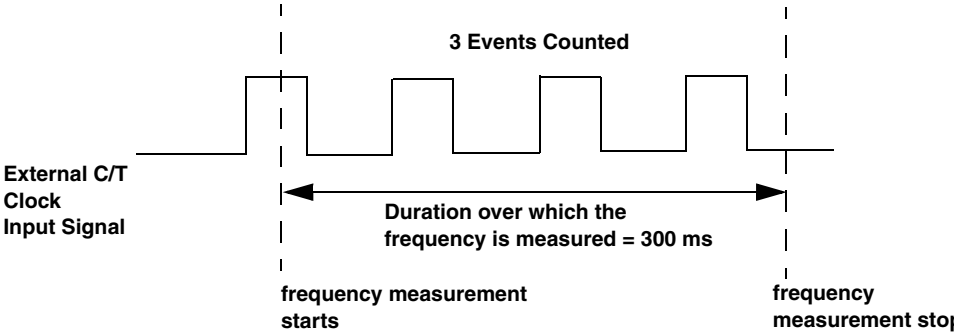


Figure 34: Example of Frequency Measurement

Using a Pulse of a Known Duration

If you need more accuracy than the Windows timer provides, you can connect a pulse of a known duration to the external gate input of a counter/timer configured for event counting; refer to the boards' user manuals for wiring details.

The following example describes how to use DTx-EZ to measure frequency using two C/T subsystems: one that generates a variable-width one-shot pulse as the gate input to a second C/T subsystem configured for event counting mode:

1. Set up one C/T subsystem for one-shot mode as follows:
   a. Use the **CTMode** function, specifying the one-shot parameter (2).
   b. For this C/T subsystem, specify the clock source (with the **ClockSource** property), the clock frequency (with the **Frequency** property if using an internal clock source or the **ClockDivider** property if using an external clock source), the gate type (with the **GateType** property), the type of output pulse (with the **PulseType** property), and the pulse width (with the **PulseWidth** property). The pulse width and period are used to determine the time that the gate is active.
   c. Configure this C/T subsystem with the **Config** method.
2. Set up another C/T subsystem for event counting mode:
   a. Use the **CTMode** property, specifying the count events parameter (0), to set up this C/T subsystem for event counting mode (and, therefore, a frequency measurement operation).
   b. For this C/T subsystem, use the **ClockSource** property to specify the clock source you wish to measure. For frequency measurement operations, an external C/T clock source is more useful than the internal C/T clock source; refer to page 154 for more information on the external C/T clock source.

c. For this C/T subsystem, use the **GateType** property to specify the gate type; ensure that the gate type for this C/T subsystem matches the active period of the output pulse from the C/T subsystem configured for one-shot mode.

d. Configure this C/T subsystem with the **Config** method.

3. Start the counter/timer configured for event counting mode with the **Start** method.

4. Start the counter/timer configured for one-shot mode with the **Start** method**.**

5. Allow a delay approximately equal to the measurement period to allow the one-shot to finish; events are counted only during the active period of the one-shot pulse.

6. For the event-counting C/T subsystem, read the number of events counted with the **CTReadEvents** method.

7. Determine the measurement period using the following equation:

$$\text{Measurement Period} = \frac{1}{\text{Actual Clock Frequency of One-Shot C/T}} * \text{Active Pulse Width of One-Shot C/T}$$

8. Determine the frequency of the clock input signal using the following equation:

$$\text{Frequency Measurement} = \frac{\text{Number of Events}}{\text{Measurement Period}}$$

Rate Generation

Use rate generation mode to generate a continuous pulse output signal from the counter; this mode is sometimes referred to as continuous pulse output or pulse train output. You can use this pulse output signal as an external clock to pace analog input, analog output, or other counter/timer operations.

To determine if the subsystem supports rate generation, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_CTMODE_RATE. If this method returns a non-zero value, the capability is supported.

To specify a rate generation mode, use the **CTMode** property, specifying the generate rate parameter (1).

Specify the C/T clock source for the operation. In rate generation mode, either the internal or external C/T clock input source is appropriate depending on your application; refer to page 154 for information on specifying the C/T clock source.

Specify the frequency of the C/T clock output signal. For an internal C/T, the **Frequency** property determines the frequency of the output pulse. For an external C/T clock source, the frequency of the clock input source divided by the clock divider (specified with the **ClockDivider** property) determines the frequency of the output pulse.

Specify the polarity of the output pulses (high-to-low transitions or low-to-high transitions) and the duty cycle of the output pulses; refer to page 162 for more information.

Also specify the gate type that enables the operation; refer to page 157 for more information on specifying the gate type.

Start rate generation mode using the **Start** method. While rate generation mode is enabled, the counter outputs a pulse of the specified type and frequency continuously. As soon as the operation is disabled, the pulse output operation stops.

To stop rate generation if it is in progress, call the **Stop**, **Abort**, or **Reset** method; **Reset** stops the operation and reinitializes the subsystem after stopping it.

Some subsystems also allow you to pause the operation using the **Pause** method and resume the paused operation using the **Continue** method. To determine if pausing is supported, use the **GetSSCaps** method, specifying the OLSSC_SUP_PAUSE capability. If this method returns a non-zero value, the capability is supported.

Figure 35 shows an example of an enabled rate generation operation using an external C/T clock source with an input frequency of 4 kHz, a clock divider of 4, a low-to-high pulse type, and a duty cycle of 50%. (The gate type does not matter for this example.) A 1 kHz square wave is the generated output.

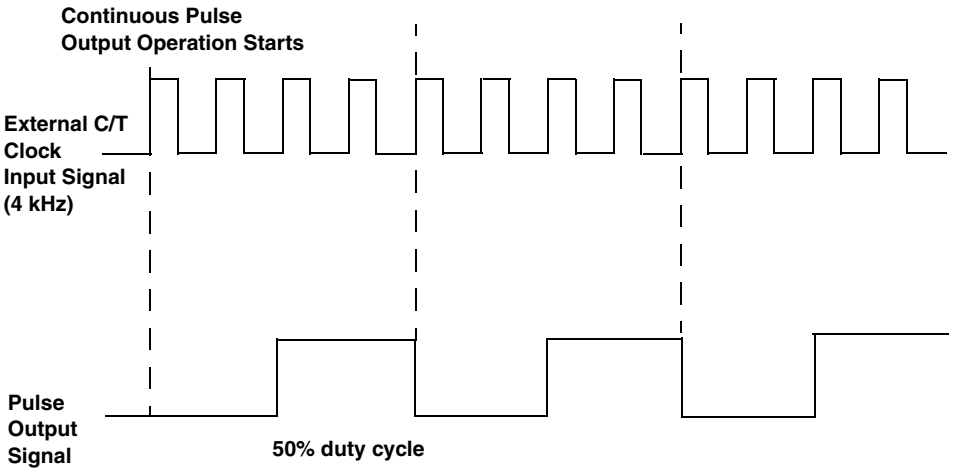


Figure 35: Example of Rate Generation Mode with a 50% Duty Cycle

Figure 36 shows the same example using a duty cycle of 75%.

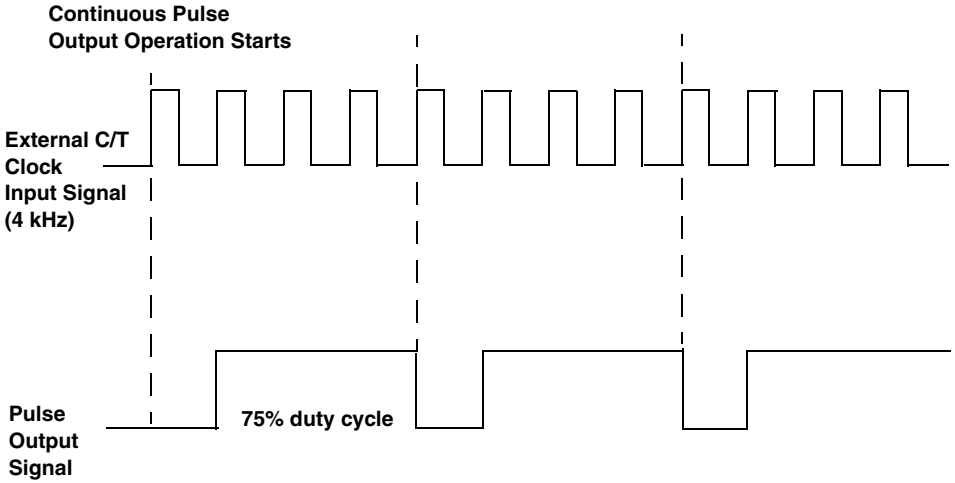


Figure 36: Example of Rate Generation Mode with a 75% Duty Cycle

Figure 37 shows the same example using a duty cycle of 25%.

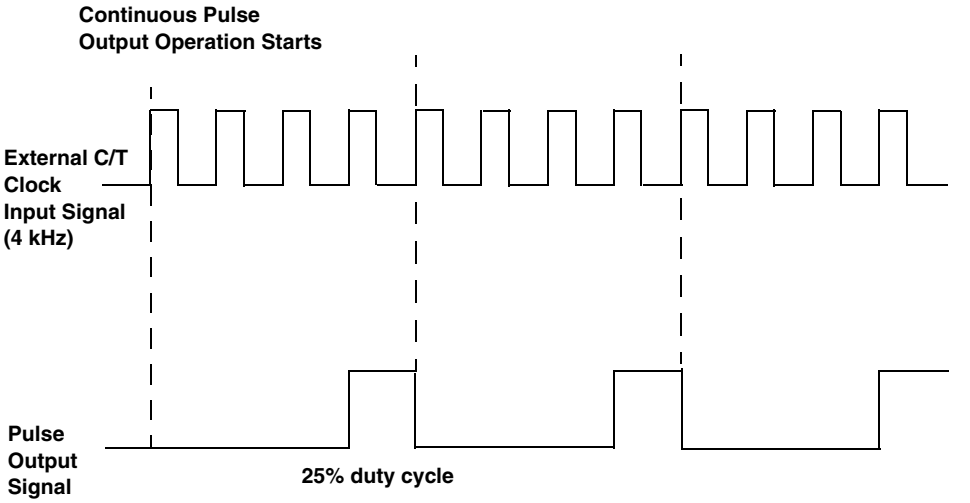


Figure 37: Example of Rate Generation Mode with a 25% Duty Cycle

One-Shot

Use one-shot mode to generate a single pulse output signal from the counter when the operation is triggered (determined by the gate input signal). You can use this pulse output signal as an external digital (TTL) trigger to start analog input, analog output, or other operations.

To determine if the subsystem supports one-shot mode, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_CTMODE_ONESHOT. If this method returns a non-zero value, the capability is supported.

To specify a one-shot operation, use the **CTMode** property, specifying the repetitive one-shot parameter (3).

Specify the C/T clock source for the operation. In one-shot mode, the internal C/T clock source is more useful than an external C/T clock source; refer to page 154 for more information on specifying the C/T clock source.

Specify the polarity of the output pulse (high-to-low transition or low-to-high transition) and the duty cycle of the output pulse; refer to page 162 for more information.

Note: In the case of a one-shot operation, use a duty cycle as close to 100% as possible to output a pulse immediately. Using a duty cycle less then 100% acts as a pulse output delay.

Also specify the gate type that triggers the operation; refer to page 157 for more information.

To start a one-shot pulse output operation, use the **Start** method. When the one-shot operation is triggered (determined by the gate input signal), a single pulse is output; then, the one-shot operation stops. All subsequent clock input signals and gate input signals are ignored.

Use software to specify the counter/timer mode as one-shot and wire the signals appropriately.

Figure 38 shows an example of a one-shot operation using an external gate input (rising edge), a clock output frequency of 1 kHz (one pulse every 1 ms), a low-to-high pulse type, and a duty cycle of 99.99%. Figure 39 shows the same example using a duty cycle of less than or equal to 1%.

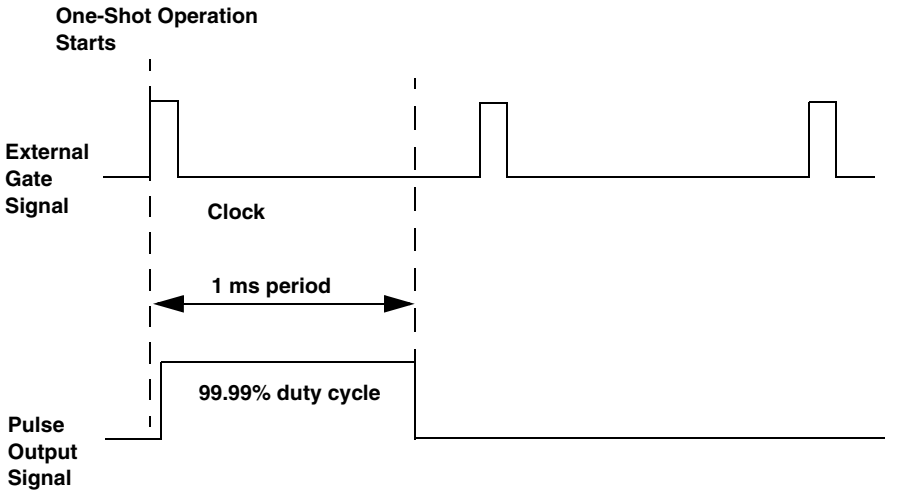


Figure 38: Example of One-Shot Mode Using a 99.99% Duty Cycle

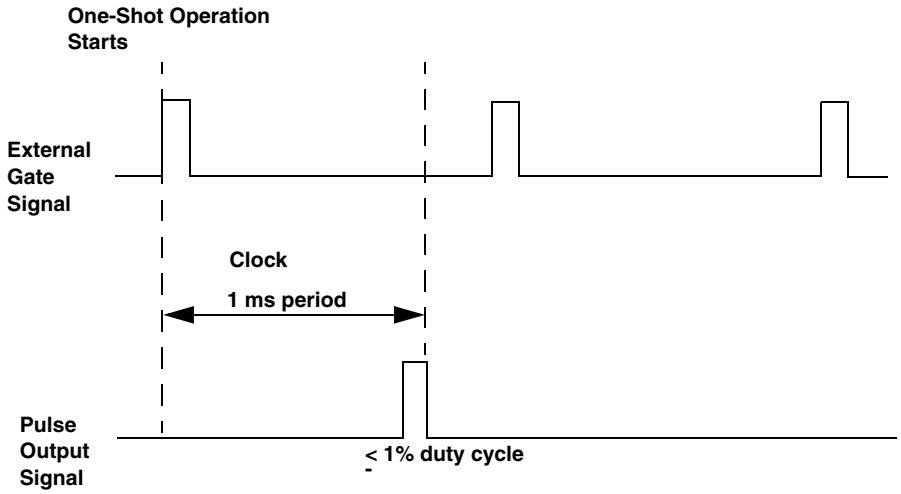


Figure 39: Example of One-Shot Mode Using a Duty Cycle Less Than or Equal to 1%

Repetitive One-Shot

Use repetitive one-shot mode to generate a pulse output signal each time the board detects a trigger (determined by the gate input signal). You can use this mode to clean up a poor clock input signal by changing its pulse width, then outputting it.

To determine if the subsystem supports repetitive one-shot mode, use the **GetSSCaps** method, specifying the capability OLSSC_SUP_CTMODE_ONESHOT_RPT. If this method returns a non-zero value, the capability is supported.

To specify a repetitive one-shot operation, use the **CTMode** property, specifying the repetitive one-shot parameter (3).

Specify the C/T clock source for the operation. In repetitive one-shot mode, the internal C/T clock source is more useful than an external C/T clock source; refer to page 154 for more information on specifying the C/T clock source.

Specify the polarity of the output pulses (high-to-low transitions or low-to-high transitions) and the duty cycle of the output pulses; refer to page 162 for more information. Also specify the gate type that triggers the operation; refer to page 157 for more information.

To start a repetitive one-shot pulse output operation, use the **Start** method. When the one-shot operation is triggered (determined by the gate input signal), a pulse is output. When the board detects the next trigger, another pulse is output.

This operation continues until you stop the operation using the **Stop**, **Abort**, or **Reset** method; **Reset** stops the operation and reinitializes the subsystem after stopping it.

Some subsystems also allow you to pause the operation using the **Pause** method and resume the paused operation using the **Continue** method. To determine if pausing is supported, use the **GetSSCaps** method, specifying the OLSSC_SUP_PAUSE capability. If this method returns a non-zero value, the capability is supported.

Note: Gates that occur while the pulse is being output may not detected by the board, depending upon your board's counter/timer. See your driver documentation for details.

Figure 40 shows an example of a repetitive one-shot operation using an external gate (rising edge); a clock output frequency of 1 kHz (one pulse every 1 ms), a low-to-high pulse type, and a duty cycle of 99.99%.

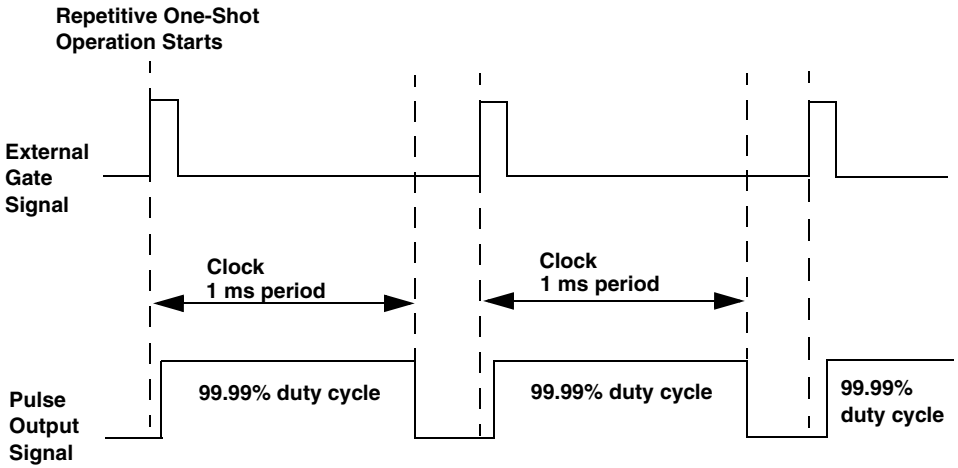


Figure 40: Example of Repetitive One-Shot Mode Using a 99.99% Duty Cycle

Figure 41 shows the same example using a duty cycle of 50%.

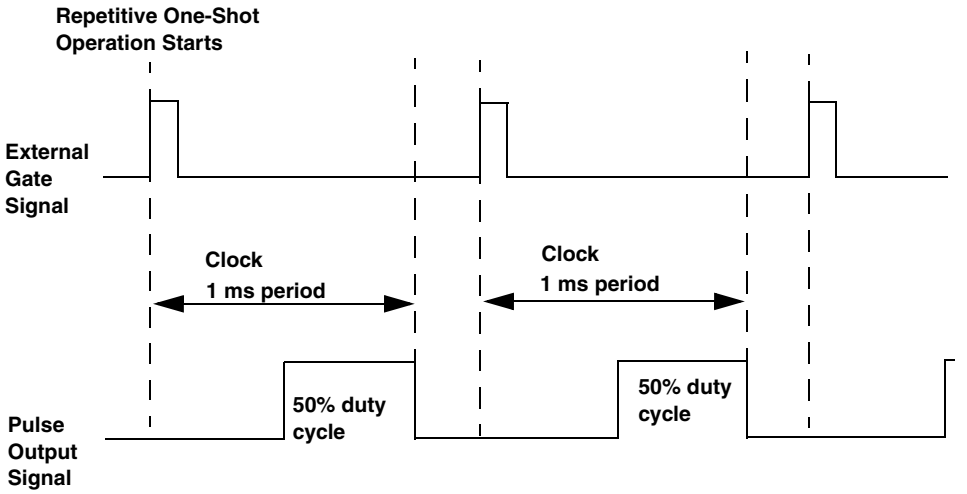


Figure 41: Example of Repetitive One-Shot Mode Using a 50% Duty Cycle

C/T Clock Sources

DTx-EZ defines the following clock sources for counter/timers:

- Internal C/T clock (this page),
- External C/T clock (page 155),
- Internally cascaded clock (page 156), and
- Extra C/T clocks (page 157).

The following subsections describe these clock sources.

Internal C/T Clock

The internal C/T clock is the clock source on the board that paces a counter/timer operation for a C/T subsystem.

To determine if the subsystem supports an internal C/T clock, use the **GetSSCaps** method, specifying the OLSSC_SUP_INTCLOCK capability. If this method returns a non-zero value, the capability is supported.

To specify the clock source, use the **ClockSource** property.

Using the **Frequency** property, specify the frequency of the clock output signal.

To determine the maximum frequency that the subsystem supports, use the **GetSSCapsEx** method, specifying the OLSSCE_MAXTHROUGHPUT capability. To determine the minimum frequency that the subsystem can produce, use the **GetSSCapsEx** method, specifying the OLSSCE_MINTHROUGHPUT capability.

External C/T Clock

The external C/T clock is a clock source attached to the board that paces counter/timer operations for a C/T subsystem. The external C/T clock is useful when you want to produce at rates not available with the internal clock or if you want to produce rates at uneven intervals.

To determine if the subsystem supports an external C/T clock, use the **GetSSCaps** method, specifying the OLSSC_SUP_EXTCLOCK capability. If this method returns a non-zero value, the capability is supported.

Specify the clock source using the **ClockSource** property. Specify the clock divider using the **ClockDivider** property; the clock input signal divided by the clock divider determines the frequency of the clock output signal.

To determine the maximum clock divider that the subsystem supports, use the **GetSSCapsEx** method, specifying the OLSSCE_MAXCLOCKDIVIDER capability. To determine the minimum clock divider that the subsystem supports, use the **GetSSCapsEx** method, specifying the OLSSCE_MINCLOCKDIVIDER capability

Internally Cascaded Clock

You can also internally connect or cascade the clock output signal from one counter/timer to the clock input signal of the next counter/timer in software. In this way, you can create a 32-bit counter out of two 16-bit counters.

To determine if the subsystem supports internal cascading, use the **GetSSCaps** method, specifying the OLSSC_SUP_CASCADING capability. If this method returns a non-zero value, the capability is supported.

Specify whether the subsystem is internally cascaded or not (single) using the **CascadeMode** property.

Note: If a counter/timer is cascaded, you specify the clock input and gate input for the first counter in the cascaded pair. For example, if counters 1 and 2 are cascaded, specify the clock input and gate input for counter 1. However, use the output from counter 2.

Extra C/T Clock Source

Extra C/T clock sources may be defined by your device driver.

To determine how many extra clock sources are supported by your subsystem, use the **GetSSCaps** method, specifying the OLSSC_NUMEXTRACLOCKS capability. Refer to your board/driver documentation for a description of these clocks.

To specify internal or external extra clock sources and their frequencies and/or clock dividers, refer to the previous subsections.

Gate Types

The active edge or level of the gate input to the counter enables or triggers counter/timer operations. DTx-EZ defines the following gate input types:

- Software (page 158),
- High-Level (page 158),
- Low-Level (page 158),
- High-Edge (page 159),
- Low-Edge (page 159),
- Any level (page 159),
- High-Level debounced (page 160),
- Low-Level debounced (page 160),
- High-Edge debounced (page 160),
- Low-Edge debounced (page 161), and
- Any level debounced (page 161).

To specify the gate type, use the **GateType** property. The following subsections describe these gate types.

Software Gate Type

A software gate type enables any specified counter/timer operation immediately when the **GateType** property is executed.

To determine if the subsystem supports a software gate, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_NONE capability. If this method returns a non-zero value, the capability is supported.

High-Level Gate Type

A high-level external gate type enables a counter/timer operation when the external gate signal is high, and disables a counter/timer operation when the external gate signal is low. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 138 for more information on these modes.

To determine if the subsystem supports a high-level external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_HIGH_LEVEL capability. If this method returns a non-zero value, the capability is supported.

Low-Level Gate Type

A low-level external gate type enables a counter/timer operation when the external gate signal is low, and disables the counter/timer operation when the external gate signal is high. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 138 for more information on these modes.

To determine if the subsystem supports a low-level external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_LOW_LEVEL capability. If this method returns a non-zero value, the capability is supported.

Low-Edge Gate Type

A low-edge external gate type triggers a counter/timer operation on the transition from the high level to the low level (falling edge). Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 151 for more information on these modes.

To determine if the subsystem supports a low-edge external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_LOW_EDGE capability. If this method returns a non-zero value, the capability is supported.

High-Edge Gate Type

A high-edge external gate type triggers a counter/timer operation on the transition from the low level to the high level (rising edge). Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 138 for more information on these modes.

To determine if the subsystem supports a high-edge external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_HIGH_EDGE capability. If this method returns a non-zero value, the capability is supported.

Any Level Gate Type

A level gate type enables a counter/timer operation on the transition to any level. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 138 for more information on these modes.

To determine if the subsystem supports a level external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_LEVEL capability. If this method returns a non-zero value, the capability is supported.

High-Level, Debounced Gate Type

A high-level, debounced gate type enables a counter/timer operation when the external gate signal is high and debounced. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 138 for more information on these modes.

To determine if the subsystem supports a high-level debounced external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_HIGH_LEVEL_DEBOUNCE capability. If this method returns a non-zero value, the capability is supported.

Low-Level, Debounced Gate Type

A low-level, debounced gate type enables a counter/timer operation when the external gate signal is low and debounced. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 138 for more information on these modes.

To determine if the subsystem supports a low-level debounced external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_LOW_LEVEL_DEBOUNCE capability. If this method returns a non-zero value, the capability is supported.

High-Edge, Debounced Gate Type

A high-edge, debounced gate type triggers a counter/timer operation on the rising edge of the external gate signal; the signal is debounced. Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 138 for more information on these modes.

To determine if the subsystem supports a high-edge debounced external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_HIGH_EDGE_DEBOUNCE capability. If this method returns a non-zero value, the capability is supported.

Low-Edge, Debounced Gate Type

A low-edge, debounced gate type triggers a counter/timer operation on the falling edge of the external gate signal; the signal is debounced. Note that this gate type is used only for one-shot and repetitive one-shot mode; refer to page 138 for more information on these modes.

To determine if the subsystem supports a low-edge debounced external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_LOW_EDGE_DEBOUNCE capability. If this method returns a non-zero value, the capability is supported.

Level, Debounced Gate Type

A level, debounced gate type enables a counter/timer operation on the transition of any level of the external gate signal; the signal is debounced. Note that this gate type is used only for event counting, frequency measurement, and rate generation; refer to page 138 for more information on these modes.

To determine if the subsystem supports a high-edge debounced external gate input, use the **GetSSCaps** method, specifying the OLSSC_SUP_GATE_LEVEL_DEBOUNCE capability. If this method returns a non-zero value, the capability is supported.

Pulse Output Types and Duty Cycles

DTx-EZ defines the following pulse output types:

- **High-to-low transitions** –The low portion of the total pulse output period is the active portion of the counter/timer clock output signal.

  To determine if the subsystem supports high-to-low transitions on the pulse output signal, use the **GetSSCaps** method, specifying the OLSSC_SUP_PLS_HIGH2LOW capability. If this method returns a non-zero value, the capability is supported.

- **Low-to-high transitions** –The high portion of the total pulse output period is the active portion of the counter/timer pulse output signal.

  To determine if the subsystem supports low-to-high transitions on the pulse output signal, use the **GetSSCaps** method, specifying the OLSSC_SUP_PLS_LOW2HIGH capability. If this method returns a non-zero value, the capability is supported.

Specify the pulse output type using the **PulseType** property.

The duty cycle (or pulse width) indicates the percentage of the total pulse output period that is active. A duty cycle of 50, then, indicates that half of the total pulse is low and half of the total pulse output is high. Specify the pulse width using the **PulseWidth** property.

Figure 42 illustrates a low-to-high pulse with a duty cycle of approximately 30%.

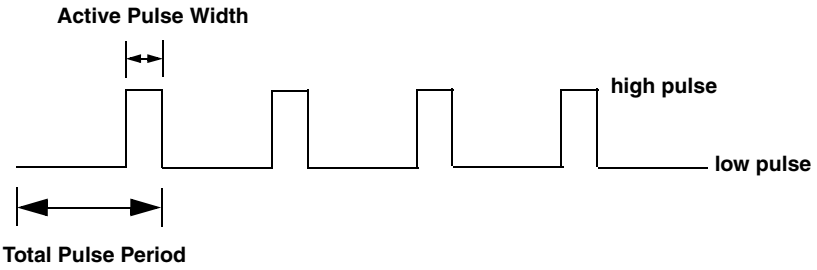


Figure 42: Example of a Low-to-High Pulse Output Type

Simultaneous Operations

If supported, you can synchronize subsystems to perform simultaneous operations. Note that you cannot perform simultaneous operations on subsystems configured for single value operations.

Note: In this section, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ name appears in italics following the Visual Basic name.

To determine if the subsystems support simultaneous operations, use the **GetSSCaps** method for each subsystem, specifying the OLSSC_SUP_SIMULTANEOUS_START capability. If this method returns a non-zero value, the capability is supported.

You can synchronize the triggers of subsystems by specifying the same trigger source for each of the subsystems that you want to start simultaneously and wiring them to the device, if appropriate.

Use the **GetSimultaneousStartList** (***olDaGetSSList***) function to allocate a simultaneous start list. Then, use the **PutSubSysOnSSList** (***olDaPutDassToSSList***) subroutine to put the subsystems that you want to start simultaneously on the start list.

Pre-start the subsystems using the **SimultaneousPreStart** (***olDaSimultaneousPreStart***) subroutine. Pre-starting a subsystem ensures a minimal delay once the subsystems are started. Once you call the **SimultaneousPreStart** (***olDaSimultaneousPreStart***) subroutine, do not alter the settings of the subsystems on the simultaneous start list.

Start the subsystems using the **SimultaneousStart** (***olDaSimultaneousStart***) subroutine. When started, both subsystems are triggered simultaneously.

Note: Do not call the **Start** method when using simultaneous start lists, since the subsystems are already started.

When you are finished with the operations, call the **ReleaseSimultaneousStartList** (***olDaReleaseSSList***) subroutine to free the simultaneous start list.

To stop the simultaneous operations, call the **Stop** (for an orderly stop), **Abort** (for an abrupt stop) or **Reset** method (for an abrupt stop that reinitializes the subsystem).

Plot Control Operations

DTx-EZ provides properties to perform the following general plotting operations:

- Plotting data (this page),
- Specifying a grid (page 168), and
- Specifying markers (page 169).

The following subsections describe these operations in more detail.

Plotting Data

When designing how your application displays data, DTx-EZ provides properties to affect the following:

- Data identification (this page),
- Plotting mechanics (page 167), and
- Appearance (page 167).

DTx-EZ provides a stripchart mode for additional flexibility as well (see page 168).

The following subsections describe these properties and the stripchart mode.

Data Identification Properties

Before you can display data, you must identify the type of information in the buffer. Use the **DataType** property to identify the data as unsigned fixed point, signed fixed point, or floating point. Use the **numChannels** property to specify the number of data channels in the buffer.

Plotting Mechanics Properties

When you plot data, you must define where in the display to start plotting data. Use the **xStart** property to set the plot's starting point.

Next, you must define the data's scale (microseconds, seconds, and so on), the length of the x-axis, and the limits of the y-axis. Use the **xAutoScale** and **yAutoScale** properties to define these parameters automatically.

If you prefer, you can manually define these parameters. To change the plot's scale, use the **xScale** property. To set the x-axis' length, use the **xLength** property. Use **yMin** to set the y-axis' lowest possible value and **yMax** to set the y-axis' highest possible value.

Appearance

You can affect the color, style, and width of the lines you use to plot data. To change a line's color, use the **Palette** property. To change a line's style (that is, dashed, dotted, and so on), use the **LineStyle** property. To change a line's width, use the **LineWidth** property.

You may also enable or disable two modes that affect how data is displayed overall if the system displays data while processing it. Set the **ForceRepaint** property to *TRUE* if you want the system to redraw the entire display each time a data element changes. Set the **UpdateMode** property to *TRUE* if you want the plot to reflect each element change.

Note: In most cases, you should set the **UpdateMode** property to *FALSE* while setting a display's parameters. After setting the parameters, change the **UpdateMode** property to *TRUE*. Doing so helps avoid errors.

Stripchart Mode

When plotting rapidly-changing continuous data, it may be easiest to use stripchart mode. Typically, the stripchart size is several times greater than the size of each buffer. When the stripchart display reaches the maximum size, the oldest data is removed from the left of the display (and internal buffer) to make room for the newest data on the right side of the plot display.

To turn stripchart mode on, set the **StripChartMode** property to *TRUE*. To set the maximum number of data points to store and display per channel, use the **StripChartSize** property.

Specifying a Grid

A grid is the underlying framework of lines on which the data is plotted. You can change the grid's appearance independently from the data's appearance. You can also choose not to display a grid at all.

You can change the grid's overall color with the **GridColor** property. You can change the grid's overall line style (dashed, dotted, and so on), with the **GridStyle** property.

Use the **GridXStart** and **GridYStart** properties to set the first line on the x- and y-axes. Use the **GridXSpacing** and the **GridYSpacing** to draw the rest of the grid lines. You can keep the x- and y-axis lines from displaying with the **GridXOn** and the **GridYOn** properties.

Finally, in response to zooming in or out of a display or to differing buffer sizes, you can choose to keep the units between the gridlines constant or to increase or decrease the space between the grid lines using the **GridAutoScale** property.

Specifying Markers

Markers are lines that you can "overlay" on a plot as reference points; they are unaffected by the plotting of data. DTx-EZ allows you to define and place up to two pairs of markers. Each pair has a horizontal and vertical line, although you may place one or two horizontal or vertical lines alone if you prefer.

To display the first pair of markers, set the **MarkerH1On** and the **MarkerV1On** properties to *TRUE*. To display the second pair, set the **MarkerH2On** and the **MarkerV2On** properties to *TRUE*.

To set the markers' positions, use the **MarkerH1Pos** and **MarkerH2Pos** properties for the horizontal markers and the **MarkerV1Pos** and the **MarkerV2Pos** properties for the vertical markers. Note that the **xScale** property affects these positions. For example, if **xScale** is set to *10*, setting **MarkerH1Pos** to *2* causes the marker to appear at the 20 unit spot. If **xScale** is set to *15*, setting **MarkerH1Pos** to *3* causes the marker to appear at the 45 unit spot.

Alternately, you can use the **MouseXPos** and **MouseYPos** properties to set the markers' positions with the mouse.

To change the markers' color, use the **MarkerColor** property. This property affects all the markers.

Finally, you can alter the data at each marker. Setting a value in the **MarkerV1Data** or **MarkerV2Data** properties alters the value of the point covered by the marker to that value specified by its respective property (**MarkerV1Data** or **MarkerV2Data**).

Product Support

General Checklist 192
Service and Support 193

General Checklist

Should you experience problems using DTx-EZ, perform the following steps:

1. Read all the appropriate sections of this manual. Make sure that you have added any "Read This First" information to your manual and that have used this information.
2. Check your distribution disk for a README file. If present, this disk will include the latest installation and configuration information.
3. Check that you have installed the device driver for your board properly.
4. Check that you have installed your hardware properly.

Note: If you are still having problems, follow the instructions provided in the next section.

Service and Support

For the latest tips, software fixes, and other product information, you can always access our World-Wide Web site at the following address: http://www.keithley.com

If you have difficulty using DTx-EZ, the Keithley Technical Support Department is available to provide technical assistance.

For the most efficient service, complete the form on page 194 and be at your computer when you call for technical support. This information helps to identify specific system and configuration-related problems and to replicate the problem in house, if necessary.

Information Required for Technical Support

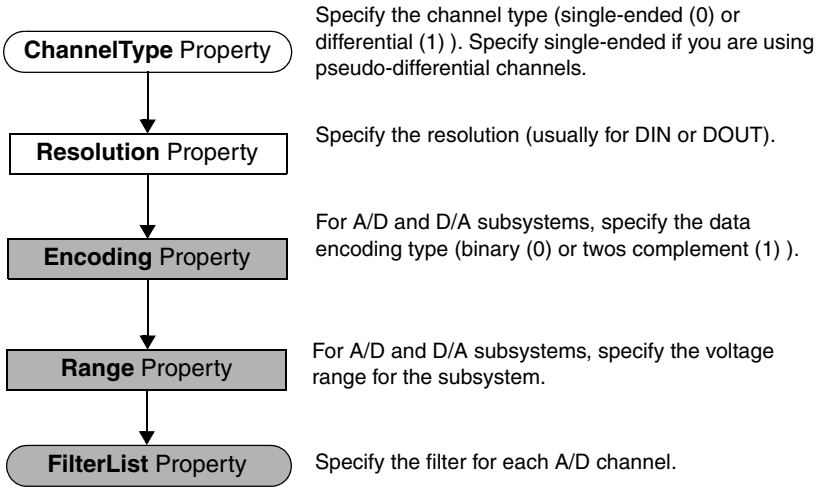
Name:_____Phone_____
Contract Number: _____
Address: _____
_____

Hardware product(s): _____
serial number: _____
configuration: _____
Device driver: _____ _____
version:_____
Software:_____ _____
serial number: _____ version:_____
PC make/model: _____
operating system: _____ version:_____
Windows version: _____
processor: _____ speed:_____
RAM: _____ hard disk space:_____
network/number of users: _____ disk cache:_____
graphics adapter: _____ data bus:_____

I have the following boards and applications installed in my system:_____
_____
_____
I am encountering the following problem(s): _____
_____
_____
_____
and have received the following error messages/codes: _____
_____
_____
I have run the board diagnostics with the following results: _____
_____
You can reproduce the problem by performing these steps:
1. _____
_____
2. _____
_____
3. _____
_____

Flowcharts for Substeps

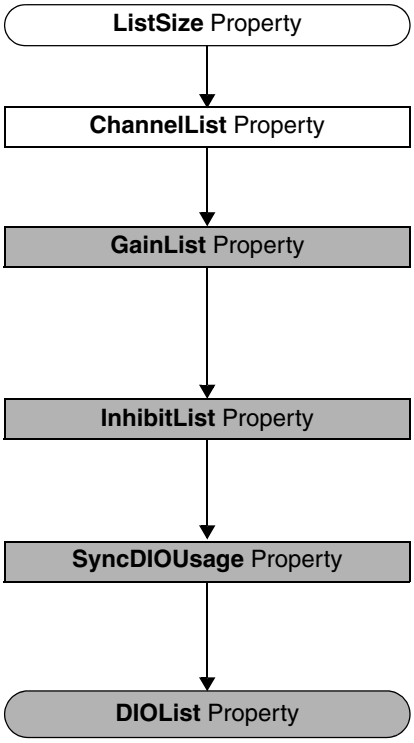
Set Subsystem Parameters


ChannelType Property
Specify the channel type (single-ended (0) or differential (1)). Specify single-ended if you are using pseudo-differential channels.
Resolution Property
Specify the resolution (usually for DIN or DOUT).
Encoding Property
For A/D and D/A subsystems, specify the data encoding type (binary (0) or twos complement (1)).
Range Property
For A/D and D/A subsystems, specify the voltage range for the subsystem.
FilterList Property
Specify the filter for each A/D channel.

Note: Depending on your board, some of these settings may not be programmable. Refer to your device driver documentation for details.

A

Set Up Channel List and Channel Parameters



Specify the size of the channel list, gain list, channel inhibit list, and synchronous digital I/O list.

Set up the channel list for the subsystem.

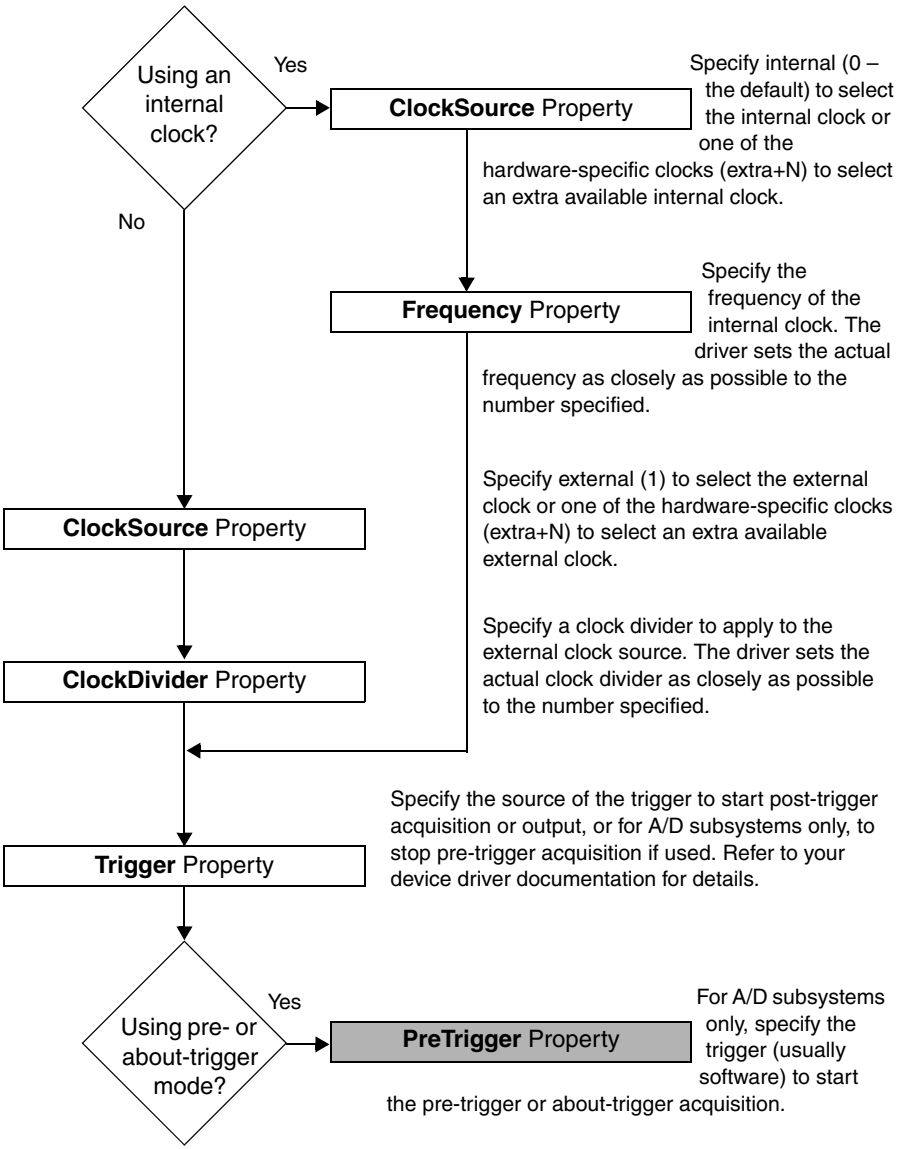
Specify the gain for each channel in the channel list (the gain list parallels the channel list). Use a gain of 1 for channels that do not support programmable gain.

Enable/disable inhibition for the specified channel entries. If inhibited, the acquired values from the specified entries are discarded.

Enable/disable a synchronous digital output operation.

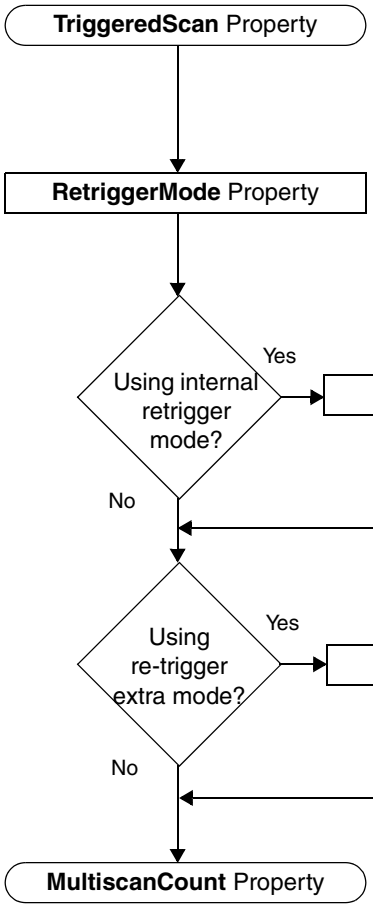
For subsystems that support synchronous digital I/O, specify the values to output to the dynamic digital output channels as each entry in the channel list is sampled.

Set Clocks, Triggers, and Pre-Triggers


Using an internal clock?
Yes
No
ClockSource Property
Specify internal (0 – the default) to select the internal clock or one of the hardware-specific clocks (extra+N) to select an extra available internal clock.
Frequency Property
Specify the frequency of the internal clock. The driver sets the actual frequency as closely as possible to the number specified.
ClockSource Property
Specify external (1) to select the external clock or one of the hardware-specific clocks (extra+N) to select an extra available external clock.
ClockDivider Property
Specify a clock divider to apply to the external clock source. The driver sets the actual clock divider as closely as possible to the number specified.
Trigger Property
Specify the source of the trigger to start post-trigger acquisition or output, or for A/D subsystems only, to stop pre-trigger acquisition if used. Refer to your device driver documentation for details.
Using pre- or about-trigger mode?
Yes
PreTrigger Property
For A/D subsystems only, specify the trigger (usually software) to start the pre-trigger or about-trigger acquisition.

A

Set Up Triggered Scan



Enable triggered scan mode.

Specify the retrigger mode: OL_RETRIGGER_INTERNAL (internal retrigger clock is the retrigger; any supported trigger source is initial trigger), OL_RETRIGGER_SCAN_PER_TRIGGER (retrigger source same as initial trigger source), or OL_RETRIGGER_EXTRA (external retrigger source is the retrigger; any supported trigger source is the initial trigger).

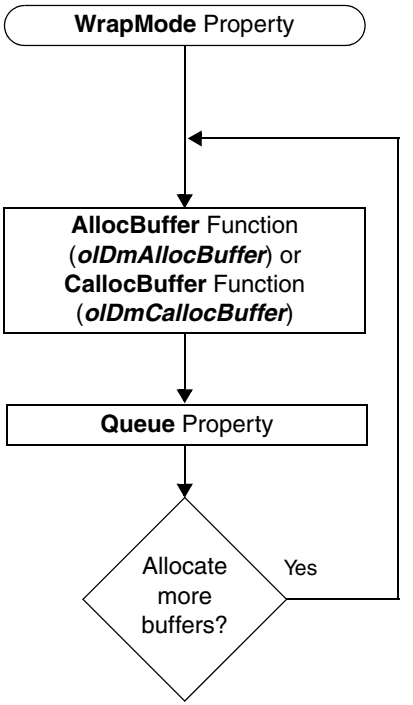
Set the frequency of the retrigger clock. The driver sets the actual frequency as closely as possible to the number specified.

Specify the retrigger source. Refer to your device driver documentation for details.

Specify the number of times to scan the channel-gain list per trigger/retrigger.

Set Up Input Buffering

In this flowchart, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ name appears in italics following the Visual Basic name.



Specify the buffer wrapping mode (if none (0), buffers are not reused if no buffers are found on the ready queue; if multiple (1), completed buffers are taken from the done queue and continuously reused when no buffers are found on the ready queue; if single (2), a single buffer is continuously reused).

Use the **AllocBuffer** function (***olDmAllocBuffer***) to allocate a buffer of samples, where each sample is 2 bytes; use the **CallocBuffer** function (***olDmCallocBuffer***) to allocate a buffer of samples of a specified size

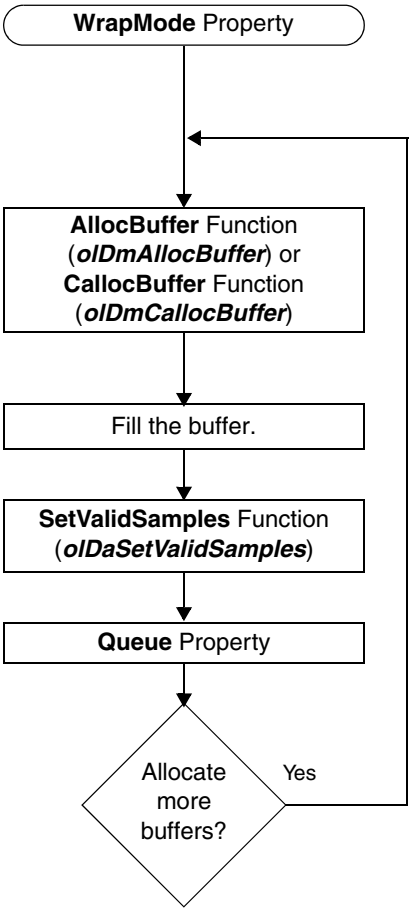
Put the buffer onto the ready queue.

A minimum of three buffers is recommended for continuous input operations if you are not using single -wrap mode.

TEquipment USA .NET
An Interworld Highway, LLC Company

Set Up Output Buffering

In this flowchart, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ name appears in italics following the Visual Basic name.



Specify the buffer wrapping mode (if none (0), buffers are not reused if no buffers are found on the ready queue; if multiple (1), completed buffers are taken from the done queue and continuously reused when no buffers are found on the ready queue; if single (2), a single buffer is continuously reused).

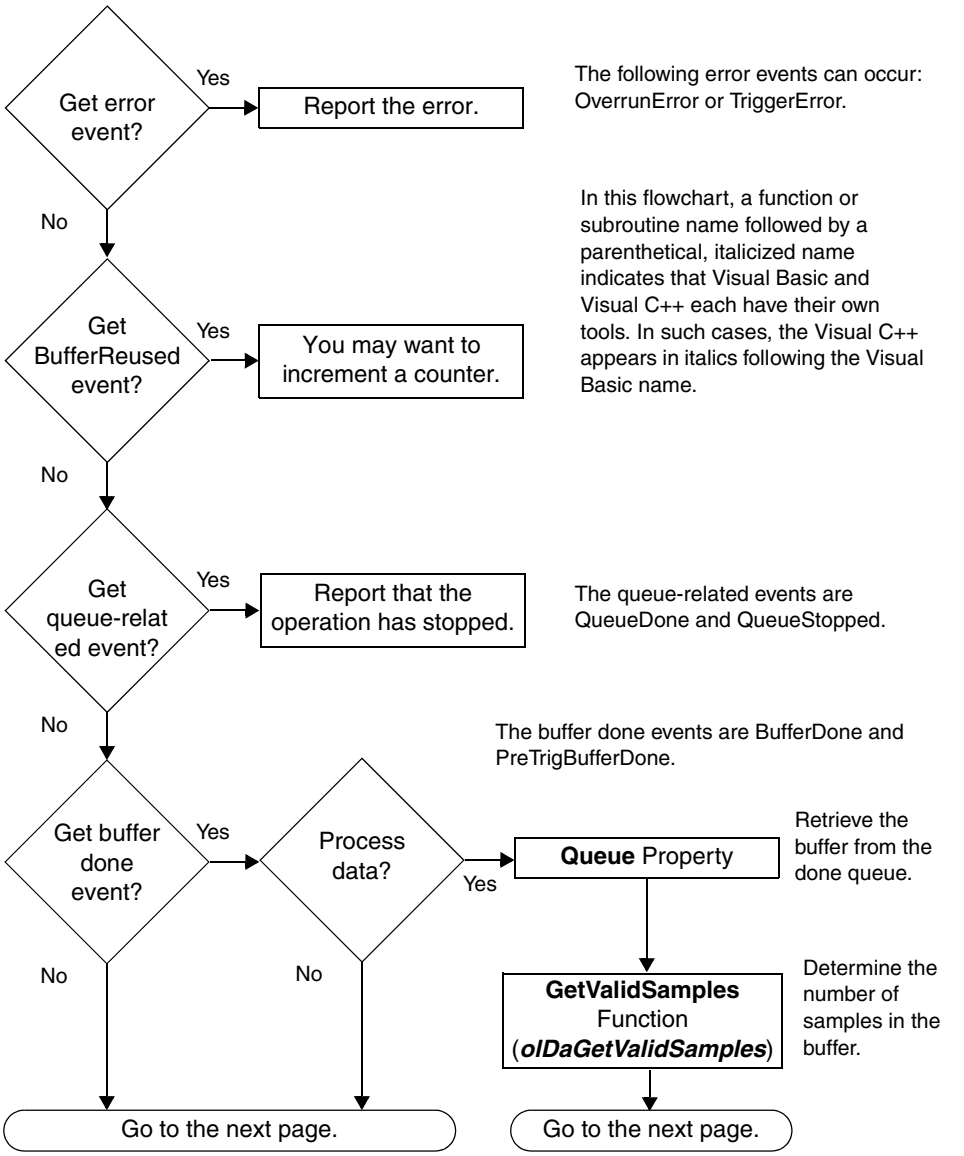
Use the **AllocBuffer** function (***olDmAllocBuffer***) to allocate a buffer of samples, where each sample is 2 bytes; use the **CallocBuffer** function (***olDmCallocBuffer***) to allocate a buffer of samples of a specified size

Specify the valid number of data points in the buffer.

Put the buffer onto the ready queue.

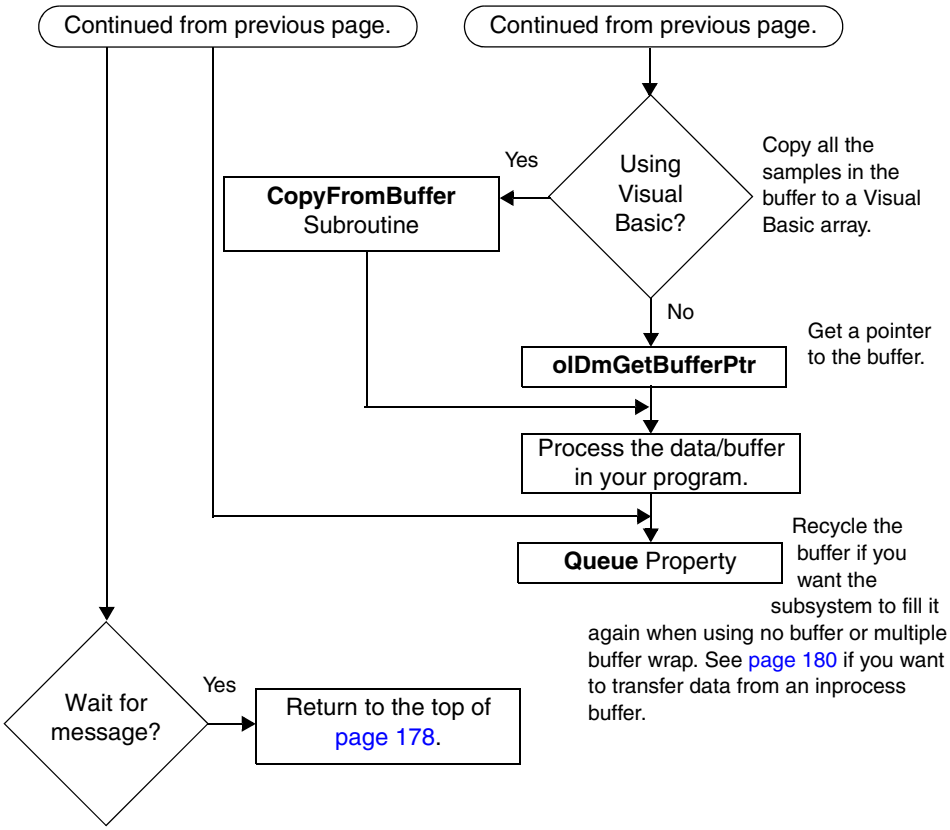
A minimum of three buffers is recommended for continuous input operations if you are not using single -wrap mode.

Deal with Events and Buffers for Input Operations


Get error event?
Yes
Report the error.
No
Get BufferReused event?
Yes
You may want to increment a counter.
No
Get queue-relat ed event?
Yes
Report that the operation has stopped.
No
Get buffer done event?
Yes
Process data?
Yes
Queue Property
No
No
GetValidSamples Function (olDaGetValidSamples)
Go to the next page.
Go to the next page.
The following error events can occur: OverrunError or TriggerError.
In this flowchart, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ appears in italics following the Visual Basic name.
The queue-related events are QueueDone and QueueStopped.
The buffer done events are BufferDone and PreTrigBufferDone.
Retrieve the buffer from the done queue.
Determine the number of samples in the buffer.

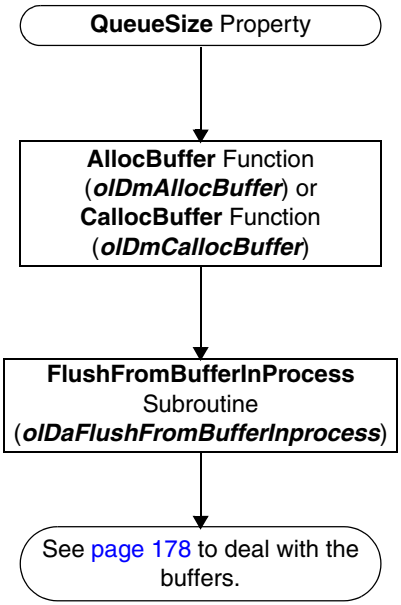
A

Deal with Events and Buffers for Input Operations (cont.)


Continued from previous page.
Continued from previous page.
Using Visual Basic?
Yes
CopyFromBuffer Subroutine
Copy all the samples in the buffer to a Visual Basic array.
No
olDmGetBufferPtr
Get a pointer to the buffer.
Process the data/buffer in your program.
Queue Property
Recycle the buffer if you want the subsystem to fill it again when using no buffer or multiple buffer wrap. See page 180 if you want to transfer data from an inprocess buffer.
Wait for message?
Yes
Return to the top of page 178.

Transfer Data from an Inprocess Buffer

In this flowchart, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ name appears in italics following the Visual Basic name.


QueueSize Property
AllocBuffer Function (***olDmAllocBuffer***) or **CallocBuffer** Function (***olDmCallocBuffer***)
FlushFromBufferInProcess Subroutine (***olDaFlushFromBufferInprocess***)
See page 178 to deal with the buffers.

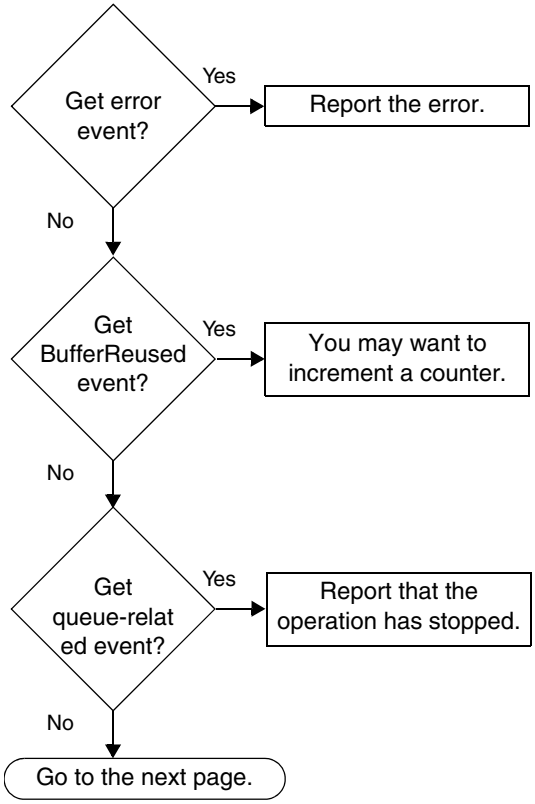
Determine the number of buffers on the inprocess queue (at least one buffer must be on the inprocess queue to perform this operation).

Use the **AllocBuffer** function (***olDmAllocBuffer***) to allocate a buffer of samples, where each sample is 2 bytes; use the **CallocBuffer** function (***olDmCallocBuffer***) to allocate a buffer of samples of a specified size

Copy the data from the inprocess buffer to the allocated buffer for immediate processing. A BufferDone event occurs when the operation completes.

A

Deal with Events and Buffers for Output Operations

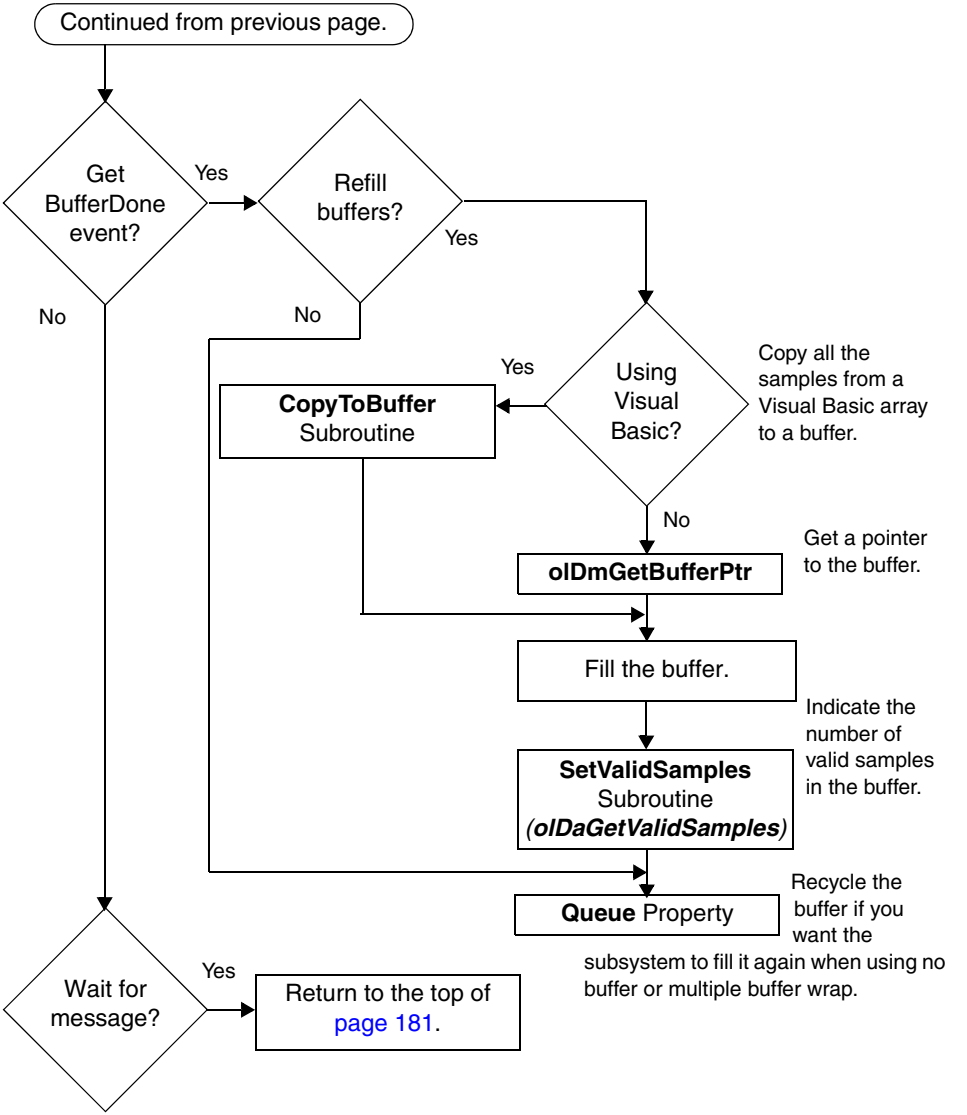


The following error events can occur: UnderrunError or TriggerError.

In this flowchart, a function or subroutine name followed by a parenthetical, italicized name indicates that Visual Basic and Visual C++ each have their own tools. In such cases, the Visual C++ appears in italics following the Visual Basic name.

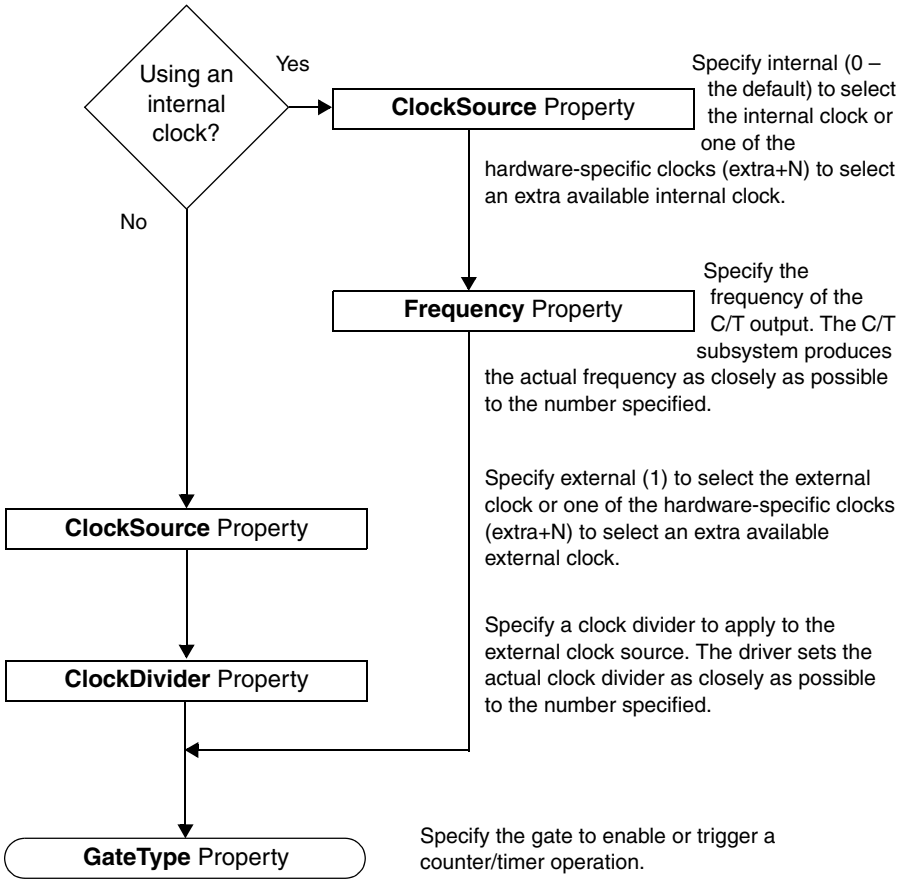
The queue-related events are QueueDone and QueueStopped.

Deal with Events and Buffers for Output Operations (cont.)

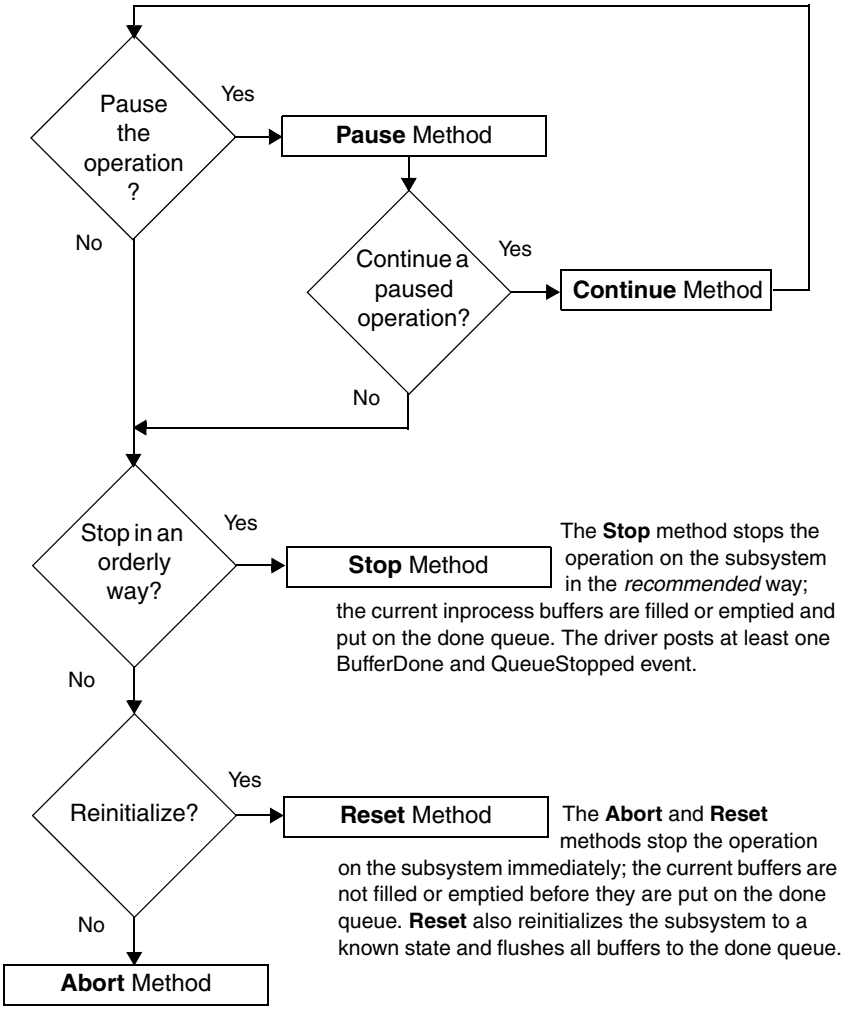

Continued from previous page.
Get BufferDone event?
Yes
No
Refill buffers?
Yes
No
Using Visual Basic?
Yes
Copy all the samples from a Visual Basic array to a buffer.
CopyToBuffer Subroutine
No
Get a pointer to the buffer.
olDmGetBufferPtr
Fill the buffer.
Indicate the number of valid samples in the buffer.
SetValidSamples Subroutine (olDaGetValidSamples)
Recycle the buffer if you want the subsystem to fill it again when using no buffer or multiple buffer wrap.
Queue Property
Wait for message?
Yes
Return to the top of page 181.

A

Set Clocks and Gates for Counter/Timer Operations


Using an internal clock?
Yes
No
ClockSource Property
Specify internal (0 – the default) to select the internal clock or one of the hardware-specific clocks (extra+N) to select an extra available internal clock.
Frequency Property
Specify the frequency of the C/T output. The C/T subsystem produces the actual frequency as closely as possible to the number specified.
ClockSource Property
Specify external (1) to select the external clock or one of the hardware-specific clocks (extra+N) to select an extra available external clock.
ClockDivider Property
Specify a clock divider to apply to the external clock source. The driver sets the actual clock divider as closely as possible to the number specified.
GateType Property
Specify the gate to enable or trigger a counter/timer operation.

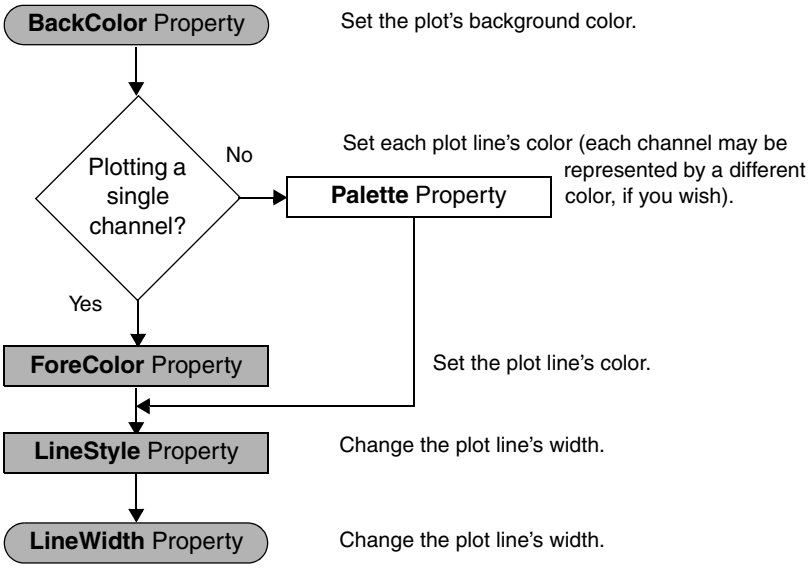
Stop the Operation



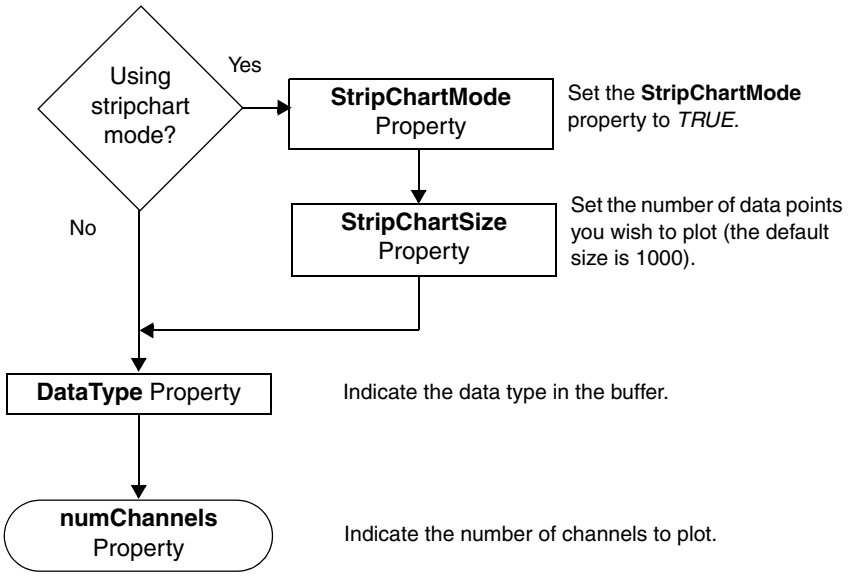
The **Stop** method stops the operation on the subsystem in the *recommended* way; the current inprocess buffers are filled or emptied and put on the done queue. The driver posts at least one BufferDone and QueueStopped event.

The **Abort** and **Reset** methods stop the operation on the subsystem immediately; the current buffers are not filled or emptied before they are put on the done queue. **Reset** also reinitializes the subsystem to a known state and flushes all buffers to the done queue.

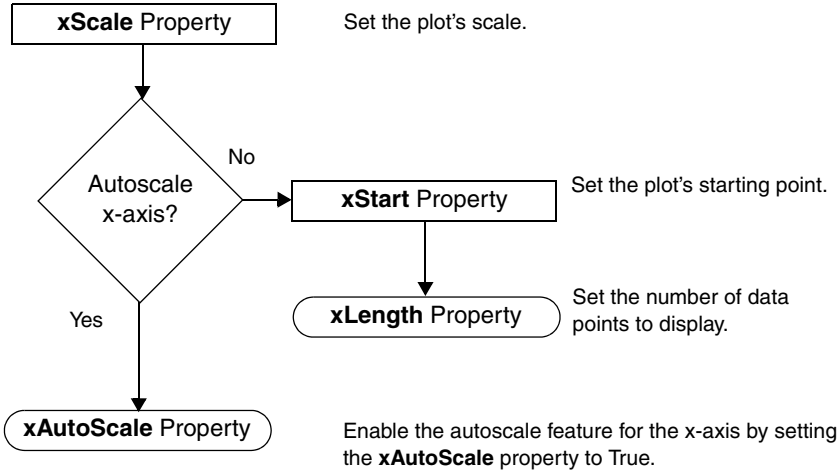
Set Plot Appearance


BackColor Property
Set the plot's background color.
Plotting a single channel?
No
Set each plot line's color (each channel may be represented by a different color, if you wish).
Palette Property
Yes
ForeColor Property
Set the plot line's color.
LineStyle Property
Change the plot line's width.
LineWidth Property
Change the plot line's width.

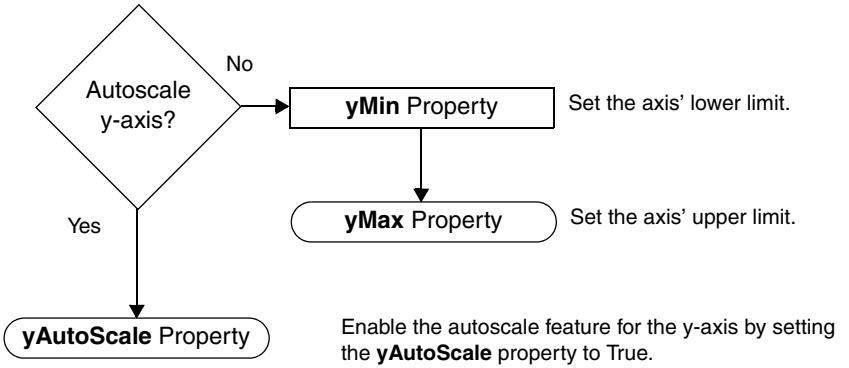
Set Pre-Operation Parameters


Using stripchart mode?
Yes
No
StripChartMode Property
Set the StripChartMode property to TRUE.
StripChartSize Property
Set the number of data points you wish to plot (the default size is 1000).
DataType Property
Indicate the data type in the buffer.
numChannels Property
Indicate the number of channels to plot.

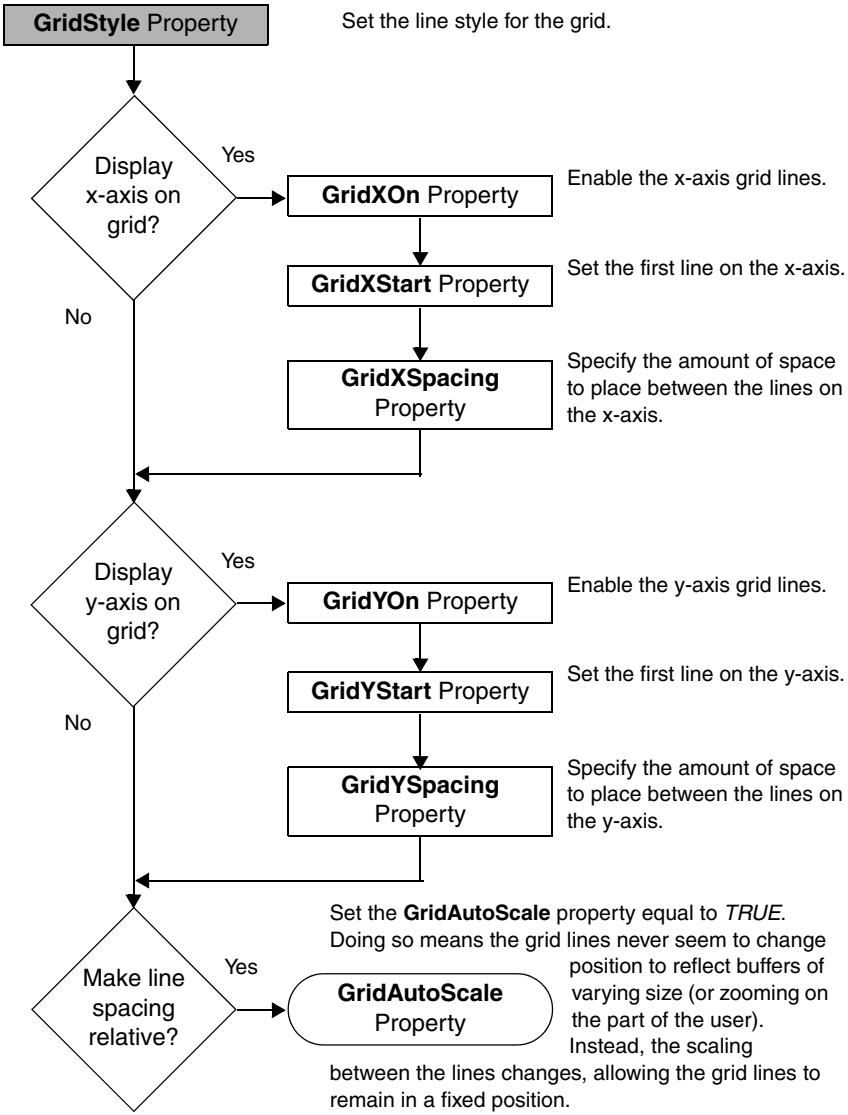
Set the Plot's x-Axis



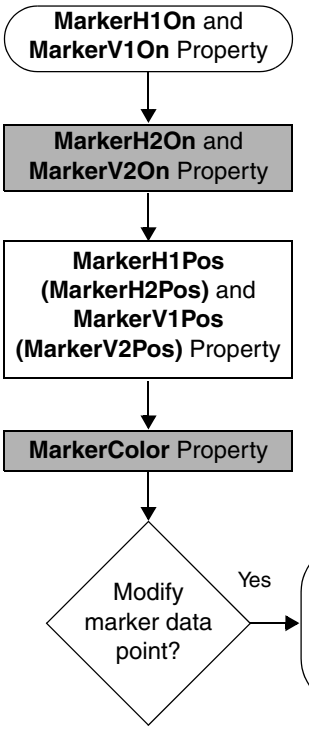
Set the Plot's y-Axis



Set Grid Parameters


GridStyle Property
Set the line style for the grid.
Display x-axis on grid?
Yes
No
GridXOn Property
Enable the x-axis grid lines.
GridXStart Property
Set the first line on the x-axis.
GridXSpacing Property
Specify the amount of space to place between the lines on the x-axis.
Display y-axis on grid?
Yes
No
GridYOn Property
Enable the y-axis grid lines.
GridYStart Property
Set the first line on the y-axis.
GridYSpacing Property
Specify the amount of space to place between the lines on the y-axis.
Make line spacing relative?
Yes
GridAutoScale Property
Set the GridAutoScale property equal to TRUE. Doing so means the grid lines never seem to change position to reflect buffers of varying size (or zooming on the part of the user). Instead, the scaling between the lines changes, allowing the grid lines to remain in a fixed position.

Set Marker Parameters



Display the horizontal (**MarkerH1On**) and vertical (**MarkerV1On**) marker lines.

Display the second horizontal (**MarkerH2On**) and vertical (**MarkerV2On**) marker lines.

Set the position of the active markers with **MarkerH1Pos** for the first horizontal marker, **MarkerH2Pos** for the second horizontal marker, **MarkerV1Pos** for the first vertical marker, and **MarkerV2Pos** for the second vertical marker.

Change the marker lines' color.

If you wish to alter the value of the data point at a marker, enter the desired value into the appropriate property. Use **MarkerH1Data** for the point at the first horizontal marker, **MarkerH2Data** for the point at the second horizontal marker, **MarkerV1Data** for the point at the first horizontal marker, and **MarkerV2Data** for the point at the second horizontal marker. (The plot must have buffer data to use these properties.)

TEquipment USA .NET
An Interworld Highway, LLC Company

Index

A

A/D
  burst example 20
  continuous example 29
abort
  a simultaneous operation 165
  an operation 184
Abort method 61, 184
  in continuous operations 113
  in event counting operations 139
  in rate generation operations 145
  in repetitive one-shot operations 152
  in simultaneous operations 165
about-trigger
  continuous operation 116
    support for 50, 117
  example using 33
allocate
  a buffer 128
  a simultaneous start list 164
AllocBuffer function 65, 128
  in inprocess buffer transfer 180
  in input buffering 176
  in output buffering 177
analog event trigger 127
  support for 55, 127
application
  creating 12
  creating a device-independent 94
asynchronous operations
  support for 51

B

BackColor property
  in setting plot appearance 185
binary encoding
  support for 54, 99
board
  initializing and specifying 93
  listing available at runtime 93
  quantity installed 93
  query for processor 57
Board property 57, 93
  in continuous buffered input operations 79
  in continuous buffered output operations 81
  in event counting operations 83
  in frequency measurement operations 85
  in pulse output operations 87
  in single-value operations 77
BoardList property 48, 76, 93
buffer 128
  allocating 128
  events 178
  identify data in for plotting 166
  multiple-wrap mode support 51, 135
  removing from done queue 133
  set up for input 176
  set up for output 177
  single-wrap mode support 51, 135
  specify quantity of channels in for plotting 166

support for 51, 128
transfer data from inprocess 180
wrap 134
write to inprocess support 51, 131
Buffer property 73
in plotting control operations 4, 89
BufferDone event 118, 131, 133
in buffer input operations 178
in buffer output operations 182
BufferReused event
in buffer input operations 178
in buffer output operations 181
with multiple buffer wrap mode 135

C

calibration
support for with software 57
CallocBuffer function 65, 128
in inprocess buffer transfer 180
in input buffering 176
in output buffering 177
CascadeMode property 60, 156
in event counting operations 83
in frequency measurement operations 85
in pulse output operations 87
cascading 156
support for 55, 156
channel 100
differential 101
quantity of differential 54, 101
quantity of DMA 51, 136
quantity of I/O 53, 100
quantity of single-ended 53, 101
set parameters 173
single-ended 101
specify gain for one or more 109
specify gain for single 109
specify one or more 102
specify single 102
specify type 101
support for filter per 54, 111
support for range per 54, 108
channel expansion 53, 100
channel-gain list
adding channels to 104
configuration example 20
determine range for entry on 109
entry inhibit support 53, 105
inhibiting entries on 105
quantity of entries on 52, 103
random support 52, 103
sequential support 52, 103
set size of 103
setting up 102, 173
specify gain for entry on 109
zero-sequential support 53, 103
ChannelList property 59, 104
in setting channel parameters 173
ChannelType property 59, 101
in setting parameters 172
ChartIt example 41
ClearError method 62
clock 154
cascading 156
external 123, 155
support for 55, 123, 155
extra 124, 157
internal 122, 155
support for 55, 122, 155
maximum internal frequency 123, 155
minimum internal frequency 123, 155

quantity of extra 55, 124, 157
set parameters for C/T operations 183
specifying 174
clock divider
maximum supported 124, 156
minimum supported 124, 156
ClockDivider property 60
in C/T operations 183
in frequency measurement operations 142
in rate generation operations 144
in setting clock parameters 174
with external clock 123, 156
ClockSource property 60, 142
example using 23
in C/T operations 183
in frequency measurement operations 140, 142
in setting clock parameters 174
with external clock 123, 156
with internal clock 122, 155
color
set grid line's 168
set maker line's 169
set plot line's 167
Config method 61, 95
in continuous buffered input operations 80
in continuous buffered output operations 82
in event counting operations 83
in frequency measurement operations 86, 142, 143
in pulse output operations 88
in single-value operations 77
configure a subsystem 95
continue an operation 184
Continue method 61, 184
in continuous operations 114
in event counting operations 139
in rate generation operations 145
in repetitive one-shot operations 152
continuous (post-trigger) mode 114
continuous A/D
example of 29
continuous about-trigger operation 116
support for 50
continuous about-trigger operation support
for 117
continuous FFT example 40
continuous operation 113
performing input 79
performing output 81
support for 50, 114
continuous pre-trigger operation 115
support for 50, 115
continuous pulse output
support for 55, 144
conventions used xiv
conversion rate
in internal retrigger mode 120
in retrigger extra mode 122
in scan-per-trigger triggered scan mode 120
CopyChannelFromBuffer subroutine 65
CopyChannelToBuffer subroutine 65
CopyFromBuffer subroutine 66, 133
in buffer input operations 179

CopyLongChannelFromBuffer
subroutine 65
CopyLongChannelToBuffer
subroutine 65
CopySingleChannelFromBuffer
subroutine 65
CopySingleChannelToBuffer
subroutine 66
CopyToBuffer subroutine 66, 133
in buffer output operations 182
counter/timer 137
cascading 156
support for 55, 156
event counting operations 138
performing 83
support for 55, 138
frequency measurement operations
support for 85, 140
high-to-low output pulse
support for 55, 162
low-to-high output pulse support 55, 162
one-shot operations 148
support for 55, 148
pulse output operations
performing 87
rate generation operations 144
support for 55, 144
repetitive one-shot operations 151
support for 55, 151
CTMode function
in frequency measurement operations 142
CTMode property 60
in event counting operations 83, 138
in frequency measurement operations 85, 140, 142
in one-shot operations 148
in pulse output operations 87
in rate generation operations 144
in repetitive one-shot operations 151
CTReadEvents method 62, 138
in event counting operations 84
in frequency measurement operations 143
custom control
adding to Visual Basic 10
adding to Visual C++ 11
Data Acquisition Custom Control 3
DT Plotting Custom Control 3
cut-off frequency
list 111

D

DAC Waveform Generator example 25
Data Acquisition Custom Control 3
adding to Visual Basic 10
adding to Visual C++ 11
data encoding 99
DataFlow property 58
in continuous (post-trigger) operations 114
in continuous about-trigger operations 117
in continuous buffered input operations 79
in continuous buffered output operations 81
in continuous pre-trigger operations 115
in single-value operations 77, 112

DataType property 70, 166
in setting plot re-operation parameters 186
DDE Server and Client example 37
DeviceName property 48
differential channel type 101
differential input
quantity of channels for 54, 101
support for 54, 101
digital event trigger 127
support for 55, 127
digital I/O
example of 35
DIOList property 59, 106
in setting channel parameters 173
divider
maximum supported 156
minimum supported 156
DMA 135
quantity of channels supported 51, 136
support for gap-free continuous operation with dual 51, 136
support for gap-free continuous operation with no 51, 136
support for gap-free continuous operation with single 51, 136
DmaUsage property 58, 136
in continuous buffered input operations 79
in continuous buffered output operations 81
done queue 131
removing buffers from 133
DT Plotting Custom Control 3
adding to Visual Basic 10
adding to Visual C++ 11
DT-Open Layers 2
DTx-EZ 2
duty cycle 162
in one-shot operations 149

E

element
defined 94
list names and types 95
list quantity of 95
selecting 94
encoding 99
binary support 54, 99
twos complement support 54, 99
Encoding property 60, 100
example using 23
in setting parameters 172
EnumBoards method 48
EnumSS method 48, 76, 95
EnumSSCaps method 49, 76
error checking 76
event 96
defined 5
respond to 178
event counting operation 138
performing 83
support for 55, 138
example
A/D burst 20
About-Trigger 33
ChartIt 41
Continuous A/D 29
Continuous FFT 40
DAC Waveform Generator 25
DDE Server and Client 37
Digital I/O 35

Scope 42
Single Value 31
external analog threshold (negative) trigger 126
support for 54, 126
external analog threshold (positive) trigger 126
support for 54, 126
external clock 123, 155
support for 55, 123, 155
external digital (TTL) trigger 125
support for 54, 125
extra clock 124, 157
quantity of 55, 124, 157
extra trigger 127
quantity supported 55, 127

F

Fast Fourier Transform
example of 40
FFT example 40
FIFO
present in data path 57
filter 111
list available during runtime 111
list cut-off frequency 111
per channel support 54, 111
quantity available during runtime 111
quantity of selections 54, 111
FilterList property 59, 111
in setting parameters 172
FilterValues property 49, 76, 111
Flush method 62, 134
FlushFromBufferInProcess subroutine
in inprocess buffer transfer 180
with inprocess queue 130
FlushFromBufferInprocess subroutine
with done queue 132
ForceRepaint property 73, 167
in plotting control operations 89
ForeColor property
in setting plot appearance 185
FreeBuffer subroutine 66, 134
frequency measurement operation 140
performing 85
support for 140
Frequency property 60
in C/T operations 183
in frequency measurement operations 142
in rate generation operations 144
in setting clock parameters 174
with internal clock 122, 155

G

gain 109
list available 109
list available during runtime 110
programmable support 53, 109
quantity available during runtime 110
quantity of selections 53, 109
specify for one or more channels 109
specify for single channel 109
GainList property 59, 110
in setting channel parameters 173
GainValues property 49, 76, 110

gap-free
  continuous operation with dual DMA support 51, 136
  continuous operation with no DMA support 51, 136
  continuous operation with single DMA support 51, 136
  input data 134
gate 157
  high-edge 159
    support for 56, 159
  high-edge debounced 160
    support for 56, 161
  high-level 158
    support for 56, 158
  high-level debounced 160
    support for 56, 160
  internal (software) 158
    support for 56, 158
  level 159
    support for 56, 159
  level debounced 161
    support for 56, 161
  low-edge 159
    support for 56, 159
  low-edge debounced 161
    support for 56, 161
  low-level 158
    support for 56, 158
  low-level debounced 160
    support for 56, 160
  set for cascaded clocks 156
  set parameters 183
GateType property 60, 157
  in C/T operations 183
  in frequency measurement operations 140, 142, 143
  with software gate 158
generate waveform
  example of 25
Get Channel Range function 59
GetBufferSize function 66
GetDataBits function 66
GetDataWidth function 66
GetDevCaps method 48, 76, 95
GetErrorString function 66
GetMaxSamples function 66
GetSimultaneousStartList function 63, 164
GetSingleValue method 61
  example using 41
  in single-value operations 78, 112
GetSSCaps method 49, 76
GetSSCapsEx method 49, 76
GetTimeDateStamp function 66
GetValidSamples function 67
  in buffer input operations 178
grid 168
  define first axes lines 168
  enable/disable relative line spacing 168
  set color 168
  set line's style 168
  set parameters 189
  show/hide lines 168
  space lines on 168
GridAutoScale property 71, 168
  in setting grid parameters 189
GridColor property 71, 168
GridStyle property 71, 168
  in setting grid parameters 189
GridXOn property 71, 168
  in setting grid parameters 189

GridXSpacing property 71, 168
in setting grid parameters 189
GridXStart property 71, 168
in setting grid parameters 189
GridYOn property 71, 168
in setting grid parameters 189
GridYSpacing property 71, 168
in setting grid parameters 189
GridYStart property 71, 168
in setting grid parameters 189

H

handle 94
hdass 94
hDass property 50
hDev property 48
help
online help 12, 13
technical support xvi
high-edge debounced gate 160
support for 56, 161
high-edge gate 159
support for 56, 159
high-level debounced gate 160
support for 56, 160
high-level gate 158
support for 56, 158
high-to-low output pulse 162
support for 55, 162

I

InhibitList property 59, 105
in setting channel parameters 173
inprocess queue 130
support for write to 51, 131
transfer data from 180
InputToVolts subroutine 68
internal (software) gate
support for 56, 158
internal clock 122, 155
support for 55, 122, 155
internal retriggered scan mode 120
support for 52, 120
internal software trigger
support for 54, 125
interrupt 135
support for 56, 135

L

LastError property 50
LastErrorDescription property 50
level debounced gate 161
support for 56, 161
level gate 159
support for 56, 159
line
enable/disable relative spacing on grid 168
set color for grid 168
set color for marker 169
set color for plot 167
set style for grid 168
set style for plot 167
set width for plot 167
show/hide on grid 168
space on grid 168
LineStyle property 70, 167
in setting plot appearance 185
LineWidth property 70, 167
in setting plot appearance 185

ListSize property 59, 103
in setting channel parameters 173
LongtoFreq (IParam) macro 86, 141
low-edge debounced gate 161
support for 56, 161
low-edge gate 159
support for 56, 159
low-level debounced gate 160
support for 56, 160
low-level gate 158
support for 56, 158
low-to-high pulse output 162
support for 55, 162

M

MagToDB subroutine 68
marker 169
alter data at 169
enable/disable 169
set line's color 169
set parameters 190
set position with code 169
set position with mouse 169
MarkerColor property 71, 169
in setting marker parameters 190
MarkerH1Data property
in setting marker parameters 190
MarkerH1On property 71, 169
in setting marker parameters 190
MarkerH1Pos property 72, 169
in setting marker parameters 190
MarkerH2Data property
in setting marker parameters 190
MarkerH2On property 72, 169
in setting marker parameters 190
MarkerH2Pos property 72, 169
in setting marker parameters 190
MarkerV1Data property 72, 169
in setting marker parameters 190
MarkerV1On property 72, 169
in setting marker parameters 190
MarkerV1Pos property 72, 169
in setting marker parameters 190
MarkerV2Data property 72, 169
in setting marker parameters 190
MarkerV2On property 72, 169
in setting marker parameters 190
MarkerV2Pos property 72, 169
in setting marker parameters 190
MaxRange property 59, 108
MaxRangeValues property 49, 76, 108
MeasureDone event
in frequency measurement
operations 86, 141
MeasureFrequency method 62, 140
in frequency measurement
operations 86
method 5
MinRange property 59, 108
MinRangeValues property 50, 76, 108
MouseXPos property 73, 169
MouseYPos property 73, 169
multiple board support 6
multiple-buffer wrap mode 135
support for 51, 135
MultiscanCount property 58
in internal retrigger mode 121
in scan-per-trigger mode 120
in triggered scan mode 119, 175

N

negative analog threshold trigger
support 54, 126
numBoards property 48, 76, 93
numChannels property 70, 166
in setting plot re-operation
parameters 186
numFilters property 49, 76, 111
numGains property 49, 76, 110
numRanges property 49, 76, 108
numResolutions property 49, 76, 100
numSubSystems property 48, 95
Nyquist Theorem 123

O

object-oriented design 6
OL_ENUM_FILTERS 111
OL_ENUM_GAINS 109
OL_ENUM_RANGES 108
OL_ENUM_RESOLUTION 100
olDaFlushFromBufferInprocess
in inprocess buffer transfer 180
with done queue 132
with inprocess queue 130
olDaGetChannelRange 59
olDaGetErrorString 66
olDaGetSSList 63, 164
olDaGetValidSamples
in buffer input operations 178
olDaGetValidSamples subroutine
in buffer output operations 182
olDaPutDassToSSList 63, 164
olDaReleaseSSList 63, 98, 165
olDaSetValidSamples
in output buffering 177
olDaSimultaneousPreStart 63, 164
olDaSimultaneousStart 63, 165
olDmAllocBuffer 65, 128
in inprocess buffer transfer 180
in input buffering 176
in output buffering 177
olDmCallocBuffer 65, 128
in inprocess buffer transfer 180
in input buffering 176
in output buffering 177
olDmFreeBuffer 66, 134
olDmGetBufferPtr 66, 133
in buffer input operations 179
in buffer output operations 182
olDmGetBufferSize 66
olDmGetDataBits 66
olDmGetDataWidth 66
olDmGetErrorString 66
olDmGetMaxSamples 66
olDmGetTimeDateStamp 66
olDmGetValidSamples 67
olDmReAllocBuffer 67
olDmReCallocBuffer 67
olDmSetValidSamples 66
olDspGetErrorString 66
olDspInputToVolts 68
olDspMagToDB 68
olDspRealFFT 68
olDspVoltsToOutput 68
olDspWindow 68
OLSSC_CGLDEPTH 52, 103
OLSSC_MAXDICHANS 54, 101
OLSSC_MAXDIGITALIOLIST_
VALUE 53, 106
OLSSC_MAXMULTISCAN 52, 119
OLSSC_MAXSECHANS 53, 101
OLSSC_NUMCHANNELS 53, 100
OLSSC_NUMDMACHANS 51, 136

OLSSC_NUMEXTRACLOCKS 55, 124, 157
OLSSC_NUMEXTRATRIGGERS 55, 127
OLSSC_NUMFILTERS 54, 111
OLSSC_NUMGAINS 53, 109
OLSSC_NUMRANGES 54, 108
OLSSC_NUMRESOLUTIONS 54, 100
OLSSC_SUP_SIMULTANEOUS_START 51, 164
OLSSC_SUP_ZEROSEQUENTIAL_CGL 53, 103
OLSSC_SUP_2SCOMP 54, 99
OLSSC_SUP_ANALOGEVENTTRIG 55, 127
OLSSC_SUP_BINARY 54, 99
OLSSC_SUP_BUFFERING 51, 128
OLSSC_SUP_CASCADING 55, 156
OLSSC_SUP_CHANNELLIST_INHIBIT 53, 105
OLSSC_SUP_CONTINUOUS 50, 114
OLSSC_SUP_CONTINUOUS_ABOUTTRIG 50, 117
OLSSC_SUP_CONTINUOUS_PRETRIG 50, 115
OLSSC_SUP_CTMODE_COUNT 55, 138, 140
OLSSC_SUP_CTMODE_ONESHOT 55, 148
OLSSC_SUP_CTMODE_ONESHOT_RPT 55, 151
OLSSC_SUP_CTMODE_RATE 55, 144
OLSSC_SUP_DIFFERENTIAL 54, 101
OLSSC_SUP_DIGITALEVENTTRIG 55, 127
OLSSC_SUP_EXP2896 53, 100
OLSSC_SUP_EXP727 53, 100
OLSSC_SUP_EXTCLOCK 55, 123, 155
OLSSC_SUP_EXTERNTRIG 54, 125
OLSSC_SUP_FIFO 57
OLSSC_SUP_FILTERPERCHAN 54, 111
OLSSC_SUP_GAPFREE_DUALDMA 51, 136
OLSSC_SUP_GAPFREE_NODMA 51, 136
OLSSC_SUP_GAPFREE_SINGLEDMA 51, 136
OLSSC_SUP_GATE_HIGH_EDGE 56, 159
OLSSC_SUP_GATE_HIGH_EDGE_DEBOUNCE 56, 161
OLSSC_SUP_GATE_HIGH_LEVEL 56, 158
OLSSC_SUP_GATE_HIGH_LEVEL_DEBOUNCE 56, 160
OLSSC_SUP_GATE_LEVEL 56, 159
OLSSC_SUP_GATE_LEVEL_DEBOUNCE 56, 161
OLSSC_SUP_GATE_LOW_ EDGE 56, 159
OLSSC_SUP_GATE_LOW_EDGE_DEBOUNCE 56, 161
OLSSC_SUP_GATE_LOW_LEVEL 56, 158
OLSSC_SUP_GATE_LOW_LEVEL_DEBOUNCE 56, 160
OLSSC_SUP_GATE_NONE 56, 158
OLSSC_SUP_INPROCESS_FLUSH 51, 131
OLSSC_SUP_INTCLOCK 55, 122, 155
OLSSC_SUP_INTERRUPT 56, 135
OLSSC_SUP_PAUSE 51
OLSSC_SUP_PLS_HIGH2LOW 55, 162

OLSSC_SUP_PLS_LOW2HIGH 55, 162
OLSSC_SUP_POSTMESSAGE 51
OLSSC_SUP_PROCESSOR 57
OLSSC_SUP_PROGRAMGAIN 53, 109
OLSSC_SUP_RANDOM_CGL 52, 103
OLSSC_SUP_RANGEPERCHANNEL 54, 108
OLSSC_SUP_RETRIGGER_EXTRA 52, 121
OLSSC_SUP_RETRIGGER_INTERNAL 52, 120
OLSSC_SUP_RETRIGGER_SCAN_PER_TRIGGER 52, 119
OLSSC_SUP_SEQUENTIAL_CGL 52, 103
OLSSC_SUP_SIMULTANEOUS_SH 53, 103
OLSSC_SUP_SINGLEENDED 53, 101
OLSSC_SUP_SINGLEVALUE 50, 112
OLSSC_SUP_SOFTTRIG 54, 125
OLSSC_SUP_SWCAL 57
OLSSC_SUP_SWRESOLUTION 54, 100
OLSSC_SUP_SYNCHRONOUS_DIGITALIO 53, 106
OLSSC_SUP_THRESHTRIGNEG 54, 126
OLSSC_SUP_TIMEREVENTTRIG 55, 127
OLSSC_SUP_TRIGSCAN 52, 118
OLSSC_SUP_WRPMULTIPLE 51, 135
OLSSC_SUP_WRPSINGLE 51, 135
OLSSCE_MAXCLOCKDIVIDER 124, 156
OLSSCE_MAXTHROUGHPUT 123, 155
OLSSCE_MINCLOCKDIVIDER 124, 156
OLSSCE_MINTHROUGHPUT 123, 155
one-shot operation 148
  repetitive support 55, 151
  support for 55, 148
online help 12, 13
operation
  stopping 98
OverrunError event
  in buffer input operations 178
  in buffer output operations 181

P

Palette property 70, 167
  in setting plot appearance 185
parameters
  set for channel 173
  set for grid 189
  set for markers 190
  set for subsystem 172
pause an operation 184
  support for 51
Pause method 61, 184
  in continuous operations 114
  in event counting operations 139
  in rate generation operations 145
  in repetitive one-shot operations 152
plot
  automatic scaling 167
  enable/disable repaint 167
  enable/disable update on each element change 167
  set appearance 185
  set line style 167

set line's color 167
set line's width 167
set pre-operation parameters 186
set scale for 167
set starting point 167
plotting control operations
performing 89
plotting data 166
positive analog threshold trigger support 54, 126
PowerOff method 62
PowerOn method 62
pre-starting subsystems 164
PreTrigBufferDone event 131
in buffer input operations 178
pre-trigger
continuous operations 115
specifying 174
support for continuous operation 50, 115
PreTrigger function
in continuous about-trigger operations 117
PreTrigger property 60, 125
in continuous pre-trigger operations 116
in setting pre-trigger parameters 174
PreTriggerBufferDone event 116, 118
processor
present on board 57
product support 191
Properties window 3
property 4
pseudo-differential analog input 102
pulse output 162
high-to-low 162
low-to-high 162
performing operation 87
pulse type
high-to-low support 55, 162
low-to-high support 55, 162
pulse width 162
PulseType property 60, 162
in frequency measurement operations 142
in pulse output operations 88
PulseWidth property 60, 162
in frequency measurement operations 142
in pulse output operations 88
PutSingleValue method 61
in single-value operations 78, 113
PutSubSysOnSSList subroutine 63, 164

Q

Queue property 62, 129, 133
in buffer input operations 178, 179
in buffer output operations 182
in input buffering 176
in output buffering 177
QueueDone event
in buffer input operations 178
in buffer output operations 181
with buffer wrap mode none 134
QueueSize property 62
in inprocess buffer transfer 180
QueueStopped event 116, 131
in buffer input operations 178
in buffer output operations 181

R

range 108
  maximum value during runtime 108
  maximum voltage range available 108
  minimum and maximum supported 108
  minimum value during runtime 108
  minimum voltage range available 108
  per channel support 54, 108
  quantity of selections 54, 108
  quantity of voltage ranges available 108
  specify 108
Range property 59, 108
  example using 23
  in setting parameters 172
rate generation operation 144
  support for 55, 144
ready queue 128
RealFFT subroutine 68
ReAllocBuffer subroutine 67
ReCallocBuffer subroutine 67
redraw 167
related documents xv
ReleaseSimultaneousStartList subroutine 63, 98, 165
repaint 167
repetitive one-shot operation 151
  support for 55, 151
requirements 8
reset
  a simultaneous operation 165
  an operation 184
Reset method 61, 184
  in continuous operations 113
  in event counting operations 139
  in rate generation operations 145
  in repetitive one-shot operations 152
  in simultaneous operations 165
resolution 100
  actual available during runtime 100
  bits of supported 100
  quantity available during runtime 100
  quantity of 54, 100
  software-programmable support 54, 100
  specify number of bits 100
Resolution property 60, 100
  in setting parameters 172
ResolutionsValues property 76
ResolutionValues property 49, 100
retrigger
  extra mode 121
  internal mode 120
  maximum scans per 52, 119
  setting the frequency for 121
ReTrigger property 60, 125
  in retrigger extra mode 122
  in triggered scan mode 175
retrigger scan
  support for extra triggered scan mode 52, 121
retriggered scan
  support for internal mode 52, 120
RetriggerFreq property 58
  in internal retrigger mode 120
  in retrigger extra mode 122
  in triggered scan mode 175
RetriggerMode property 58
  in internal retrigger mode 120
  in retrigger extra mode 121

in scan-per-trigger mode 119
in triggered scan mode 175

S

sampling theory 123
scale
automatically for plots 167
setting 167
scan
internal retrigger mode support 52, 120
maximum per trigger or retrigger 52, 119
multiple on trigger 118
retrigger-extra mode support 52, 121
setting up 175
triggered support 52, 118
scan-per-trigger triggered scan mode
support for 52, 119
Scope example 42
service and support procedure 193
SetValidSamples function 66
in output buffering 177
SetValidSamples subroutine
in buffer output operations 182
simultaneous operation 164
simultaneous sample-and-hold 103, 107
support for 53, 103
simultaneous start
allocating list 164
ending operation 165
free list 165
pre-starting subsystems on list 164
put subsystem on list 164
releasing list 98
starting subsystems on list 165
support for 51, 164
SimultaneousPreStart subroutine 63, 164
SimultaneousStart subroutine 63, 165
single value
example of 31
single-buffer wrap mode
support for 51, 135
single-ended
quantity of channels 53, 101
single-ended channel input 101
support for 53, 101
single-ended channel type 101
SinglePoint property 73
example using 41
in plotting control operations 89
single-value operation 112
performing 77
support for 50, 112
software (internal) trigger 125
software gate 158
SRL subsystem
restrictions on 94
start a simultaneous operation 165
Start method 61
example using 24, 27, 41
in continuous buffered input operations 80
in continuous buffered output operations 82
in continuous operations 113
in event counting operations 84, 138
in frequency measurement operations 143
in one-shot operations 149
in pulse output operations 88

in rate generation operations 145
in repetitive one-shot operations 152
starting point
set for plot 167
stop
a simultaneous operation 165
an operation 98, 184
Stop method 61, 98, 184
example using 24, 27, 41
in continuous operations 113
in event counting operations 139
in rate generation operations 145
in repetitive one-shot operations 152
in simultaneous operations 165
stripchart mode 168
enable/disable 168
example using 27, 33, 41
StripChartMode property 70, 168
in setting plot re-operation parameters 186
StripChartSize property 70, 168
in setting plot re-operation parameters 186
style
set grid line's 168
set plot line's 167
SubSysElement property 57, 94
in continuous buffered input operations 79
in continuous buffered output operations 81
in event counting operations 83
in frequency measurement operations 85
in pulse output operations 87
in single-value operations 77
subsystem
configuring 95
defined 94
pre-starting 164
put on simultaneous start list 164
quantity available 95
set parameters 172
subsystem handle 94
SubSystem property 57, 94
in continuous buffered input operations 79
in continuous buffered output operations 81
in event counting operations 83
in frequency measurement operations 85
in pulse output operations 87
in single-value operations 77
SubSystemList property 48, 95
SubSysType property 57, 94
in continuous buffered input operations 79
in continuous buffered output operations 81
in event counting operations 83
in frequency measurement operations 85
in pulse output operations 87
in single-value operations 77
support 191
SyncDIOUsage property 59, 106
in setting channel parameters 173
synchronous digital I/O
setting up a channel-gain list for 106
synchronous digital output channel list
maximum value 53, 106

synchronous digital output operation
support for 53, 106

T

technical support xvi, 191, 193
timer event trigger 127
support for 55, 127
trigger
acquire data before 115
acquire data before and after 116
acquire or output data continuously on 114
analog event 127
support for 55, 127
digital event 127
support for 55, 127
external analog threshold (negative) 126
support for 54, 126
external analog threshold (positive) 126
support for 54, 126
external digital (TTL) 125
support for 54, 125
extra 127
internal retrigger mode 120
maximum scans per 52, 119
quantity of extra 55, 127
restrictions 152
retrigger extra mode 121
scan CGL multiple times on 118
scan-per-trigger triggered scan mode 119
software (internal) 125
support for 54, 125
specifying 174
synchronizing 164
timer event 127
support for 55, 127
Trigger property 60, 125
example using 23
in continuous (post-trigger) mode 114
in continuous about-trigger mode 117
in continuous pre-trigger mode 116
in setting trigger parameters 174
triggered scan 118
setting up 175
support for 52, 118
support for scan-per-trigger mode 52, 119
TriggeredScan property 58, 118
example using 24
in setting up triggered scan 175
TriggerError event 120
in buffer input operations 178
in buffer output operations 181
in retrigger extra mode 122
troubleshooting
service and support procedure 193
troubleshooting checklist 192
twos complement encoding
support for 54, 99

U

UpdateMode property 73, 167
in plotting control operations 89

V

ValueToVolts function 68

Visual Basic project
  adding custom controls to 10
Visual C++ project
  adding custom controls to 11
VoltsToOutput subroutine 68
VoltsToValue function 68

W

waveform
  example of generating 25
Waveform Generator example 39
what you need 8
width
  set plot line's 167
Window subroutine 68
Windows timer
  using 140
wrap mode 134
  multiple support 51, 135
  single support 51, 135
WrapMode property 58
  in input buffering 176
  in output buffering 177
  with buffer wrap mode none 134
  with multiple buffer wrap mode 135
  with single buffer wrap mode 135

X

xAutoScale property 72, 167
x-axis
  automatic scaling 167
  set first line on 168
  set length of 167
  show/hide lines on 168
xLength property 72, 167
xScale property 72, 167, 169
  in setting plot x-axis 187
xStart property 72, 167

Y

yAutoScale property 73, 167
  in setting plot y-axis 188
y-axis
  automatic scaling 167
  define upper and lower limits 167
  set first line on 168
  show/hide lines on 168
yMax property 73, 167
  in setting plot y-axis 188
yMin property 73, 167
  in setting plot y-axis 188

Z

zooming, example of 28

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments, Inc.
All other trademarks and trade names are the property of their respective companies.

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY (534-8453) • www.keithley.com

12/04